Design Time Methodology for the Formal Modeling and Verification of Smart Environments

POLITECNICO DI TORINO

SCUOLA DI DOTTORATO

Dottorato in Ingegneria Informatica e del Sistemi – XXV ciclo

Tesi di Dottorato

Design Time Methodology for the Formal Modeling and Verification of Smart Environments

Muhammad Sanaullah

Tutore
Prof. Fulvio Corno

Coordinatore del corso di dottorato
Prof. Pietro Laface

February 2014

to my beloved uncle
Sheikh Ikram Shahzad
(late)
and my most respected
teacher
Hazrat Sayad
Muhammad
Bader-ul-Zaman Shah sb.
(late)

Acknowledgements

"In the name of GOD, the Most Gracious and the Most Merciful"

I would like to express special thanks to my Ph.D. advisor Professor Fulvio Corno, whose encouragement, inspiration, great efforts in welcoming the new ideas and elaborating them clearly helped me think as a research scientist. His sincere guidance, knowledge and vision assisted me at every step of the doctoral program and ensured the completion of my dissertation.

I would like to thank e-Lite research group members, especially Dr. Dario Bonino and Dr. Faisal Razzak, for giving me time to discuss various issues related to my research activities. I would like to thank the faculty, staff and technicians of Dipartimento di Automatica ed Informatica (DAUIN), Scuola Di Dottorato (SCUDO) and the Politecnico Di Torino – Italy, whose active involvement facilitated me in every matter.

I am very thankful to my funding agency Higher Education Commission (HEC) Pakistan, who gave me the opportunity to attain the highest degree in education from abroad.

I am grateful to Dr. Muhammad Jamaluddin Thaheem who always delightfully accepts my work for review and provides detailed feedbacks. I would like to say thanks to the anonymous reviewers of different conferences and journals whose comments helped in purifying and shaping my research in a more constructive and convincing way. I am grateful to the examiner committee, Professor Paolo Montuschi, Professor Filippo Lanubile and Professor Alberto Bosio, who will give time from their busy schedule to review my dissertation.

I feel great pleasure to extend my thanks to my parents, uncles, brothers, sisters and in-laws. Their wishes, efforts, prayers, love and care have always been a source of motivation for me.

At the end, I have no words to express the appreciation to my beloved wife Mina Arshad who always understands and encourages me at every moment.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

> Smart environment is "a physical world that is richly and invisibly interwoven with sensors, actuators, displays, and computational elements, embedded seamlessly in the everyday objects of our lives, and connected through a continuous network."
>
> *Mark Weiser* [1]

> Smart environment is "a small world where different kinds of smart device are continuously working to make inhabitants' lives more comfortable."
>
> *Diane J. Cook and Sajal K. Das* [2]

Smart Environments (SmE) are digital worlds which are highly sensitive, adaptive and responsive with the users' activities [3,4]. The roots of SmE come from multi-disciplinary fields; computer science, electrical engineering, industrial design, human-computer interaction and cognitive sciences [5]. Thanks to the advancements in sensor functionalities, artificial intelligence, ubiquitous and pervasive computing technologies, SmE have gained the capacity to deliver various services in intelligent manner by considering the presence and actions of users [1, 3, 6]. For specific services, users can interact with the system in any manner and at any time. The system consists of various heterogeneous devices, which range from simple sensors to multi-feature devices, and participate to achieve desired services.

The basic objective of SmE is to provide intelligent services, such as energy management, temperature management, assisted living or theft prevention [7–9]. The safety and security services are also some of the essential requirements for many SmE, and depend upon the context (interchangeably mentioned as 'environment') and domains of the application [4, 10]. For example, the safety service in case of fire is to switch on the security alarms, unlock and open the emergency exit doors, turn on the emergency and path-pointing lights directing people towards the emergency exit, make recorded calls to the

nearby fire and rescue offices and other key officials of respective environment; whereas the security requirement for accessing the bank is achieved by crossing two automatically locked doors in which one door will not open until the other is closed.

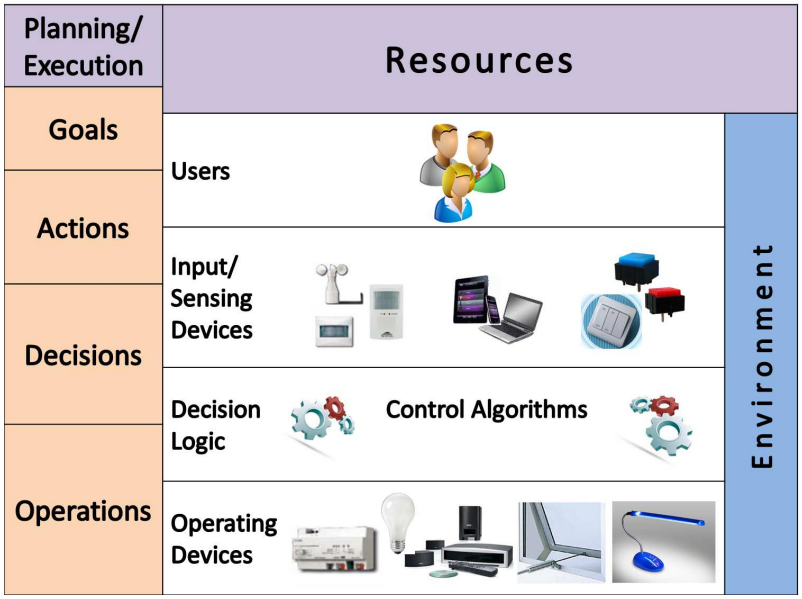

Figure 1.1. A Framework for Smart Environments

For providing the services automatically and in intelligent way, some computation is added through which the functionalities of these associated heterogeneous devices can be controlled [11, 12]. These computational components are also known as Control Algorithms and use artificial intelligence (e.g. fuzzy logics, decision trees, machine learning, case-based reasoning and temporal reasoning) or database (e.g. event-condition-action rules and SQL-based data management) techniques for decision making in an intelligence manner.

The basic elements of SmE are users, devices, control algorithms and environment, as represented in Figure 1.1. With their interactions, the desired functionalities can be achieved. A generic execution flow in SmE can be classified in four layers: goals, actions, decisions and operations. Goals are the desires of the users which they want to achieve from SmE (such as turn on the TV, turn off all the lights except those of TV lounge, switch the home in sleeping mood, etc.). For achieving a certain goal, users have to perform some specific action. The actions can be sensed through sensors or they can be input by directly performing them on the devices, or can also be commanded by using the designed APIs of the SmE (through various handheld devices).

When user performs any action, a notification message (or a set of messages) is sent to the control algorithms. Control algorithms reside at gateway levels, where the concerning devices are also connected by using some wired or wireless medium, and the requirements

related to the safety, security and reliable behavior of SmE are incorporated and enforced through them. Control algorithms act as sophisticated bridge between the input actions and the output operations. Against each incoming message, the current configuration of the system and devices is considered, and according to the incorporated constraints, a decision for the specific operations (services) is made. Further, on the basis of these decisions, control algorithms send the relevant commands to the devices for performing the decided operation. The devices, according to their current configuration and internal constraints, perform the specific operations and acknowledge back about the status of the operation to the control algorithms (these acknowledgments are also considered as notifications).

By having these sophisticated controlling features, SmE are currently being introduced in homes, hospitals, offices, industries, airports, railways, transportation mediums and many other important (industrial and public) places [4]. On the basis of adopted technologies and their various application scenarios, in literature, such environments are mostly referred as Smart Environments (SmE), Smart Space, Intelligent Environments (IEs), Ambient Intelligence (AmI), Smart Home and Intelligent Domotic Environments (IDE).

1.1 Motivation

A service offered by SmE can consist (of controlling the functionalities) of more than one device. The devices are of heterogeneous nature and self-independent with their own working functionalities and internal behaviors. For achieving the desired functionalities, some specific relevant commands are required to post on these devices. For a night mood service, as depicted in Figure 1.2, in which various devices, such as windows, window shutters, lights, doors, burglar alarm and many other, are required to be controlled, and each device accepts its relevant commands.

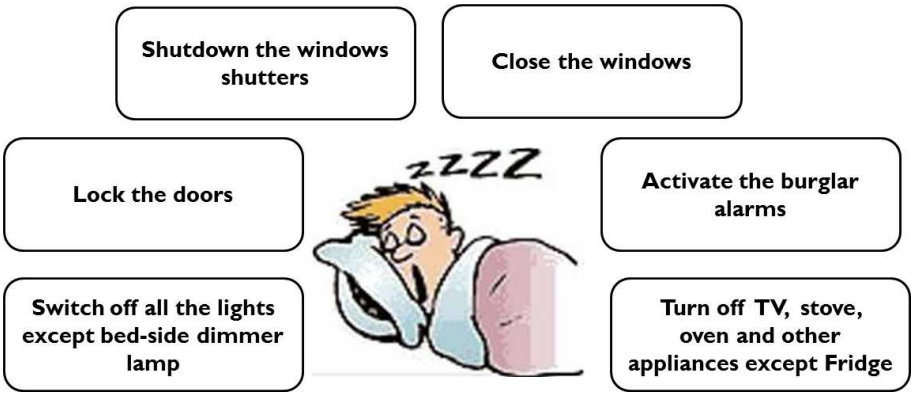

Figure 1.2. Night Mood: A Service

The devices which are used in SmE can also be complex by having multiple features, such as a service can be requested against which the TV is required to be switched “ON” with channel “22”, volume “60”, brightness “70”, contrast “60” and sharpness “50”. Depending upon the current configuration of the TV, different sets of commands can be posted (the commands when the TV is in “OFF” state are different from the commands when it is in “ON” state). Further, the system level constraints (related to the overall safety, security and reliable behavior) add innumerous details in the system. Due the their control (of heterogeneous devices), various system level constraints, distributed and parallel state information, synchronous and asynchronous interactions, multiple sources of control (as one device can be associated with more than one services) and different evolution paths (depending upon the current configurations), the behavior of SmE becomes difficult to predict (by the designers of SmE) and may even lead to error-prone scenarios.

SmE systems can be implemented in sensitive environments (like fire control units, theft or traffic control systems, nursing care houses and others) where the occurrence of errors may cause critical/unwanted situations. For avoiding the errors, their thorough verification is very important. The verification process can be performed at design or implementation time. Based on the complexities and their application scenarios, it is advisable to verify SmE models at design time for reducing criticality, time, cost and energy, and achieving reliability [13–16].

For the design time verification, simulation or formal (mathematical) methods (strategies and structured approaches) are commonly used with their own strengths and limitations. As complexity and ambiguity are usually the common features of such systems, and it may not be effective to verify the accuracy on all possibly reachable paths through simulation [17, 18]. Therefore a technique is required which may ensure the exhaustive verification of various requirements. Thus, the use of formal methods helps to root them out, and, in result, a reliable secure system can be designed which has all the desired features and consistency among its integrated components with the environment [14]. Moreover, formal methods promise holistic design time verification based on the following strengths:

- they are strongly based on mathematical evidence and increase the understandability of the modeled system;
- they are used for reliably modeling a system at design time;
- they can model the concerning requirements in the form of properties by using logic based on mathematics;
- they can formally verify the modeled system against the requirements (reliable behavior, along with other requirements of the system);
- they can trace back the errors and can help in fixing them at early design stages.

1.2 Problem Statement

The SmE are the integration of hardware (devices) and software (control algorithms) components which continuously interact with each other in a requirement-accomplishing manner according to the presences, locations, actions or behaviors of the user. The software components have the information of the imposed constraints and some form of intellectual strategies for automatically controlling the functionalities of various devices.

The systems are huge as a number of heterogeneous devices are introduced with added functionalities. For automatically controlling the functionalities of each devices (as the manufactures are also many), their associated commands are required for triggering them. Their heterogeneous nature and complex parallel and hierarchical behaviour (depending upon the offered functionalities), may introduce ambiguity. Similarly, for obtaining more sophistication, various system level constraints are progressively added in SmE specifications, which also introduce complexity and ambiguity for controlling the SmE. The wishes for facilitating the users at various levels, by considering their actions and behaviors, according to their context, also become a reason of increasing complexity.

The impact for these complexities at individual and collectively controlled level may not allow the designers, developers or programmers to confidently claim about the correctness, completeness and consistence reliable behaviour of SmE under all circumstances. As the applicability of SmE is also in sensitive fields where any wrong decision or the existence of bug may lead to unwanted or critical situation, therefor the reliance of these systems strongly demands confirmations.

The confirmation of correctness, completeness and consistent behaviour regarding the specifications is an essential concern of SmE stakeholders (designers, developers, programmers as well as their users) and can be satisfied when the system is designed and verified by adopting a suitable methodology: starting from lower level details to higher level goals achievements.

1.3 Contribution

The major objective of this thesis is to resolve the incomplete, incorrect and inconsistent behavioral issues of SmE. For ensuring the solution of these issue, a design time methodology is proposed, which provides the guideline to the SmE designers and validators by addressing and revaluing the important concerns and factors which are required to be considered during the modeling and verification of SmE. Formal method techniques are adopted for the modeling and verification purposes, due to their several advantages.

The process starts from the lower level formal modeling and verification of each component to the higher level formal modelling and achievement of goals. A set of technologies are designed and developed by following the proposed guidelines. Different case studies and experiments are conducted, the results ensure the reliability and adaptability

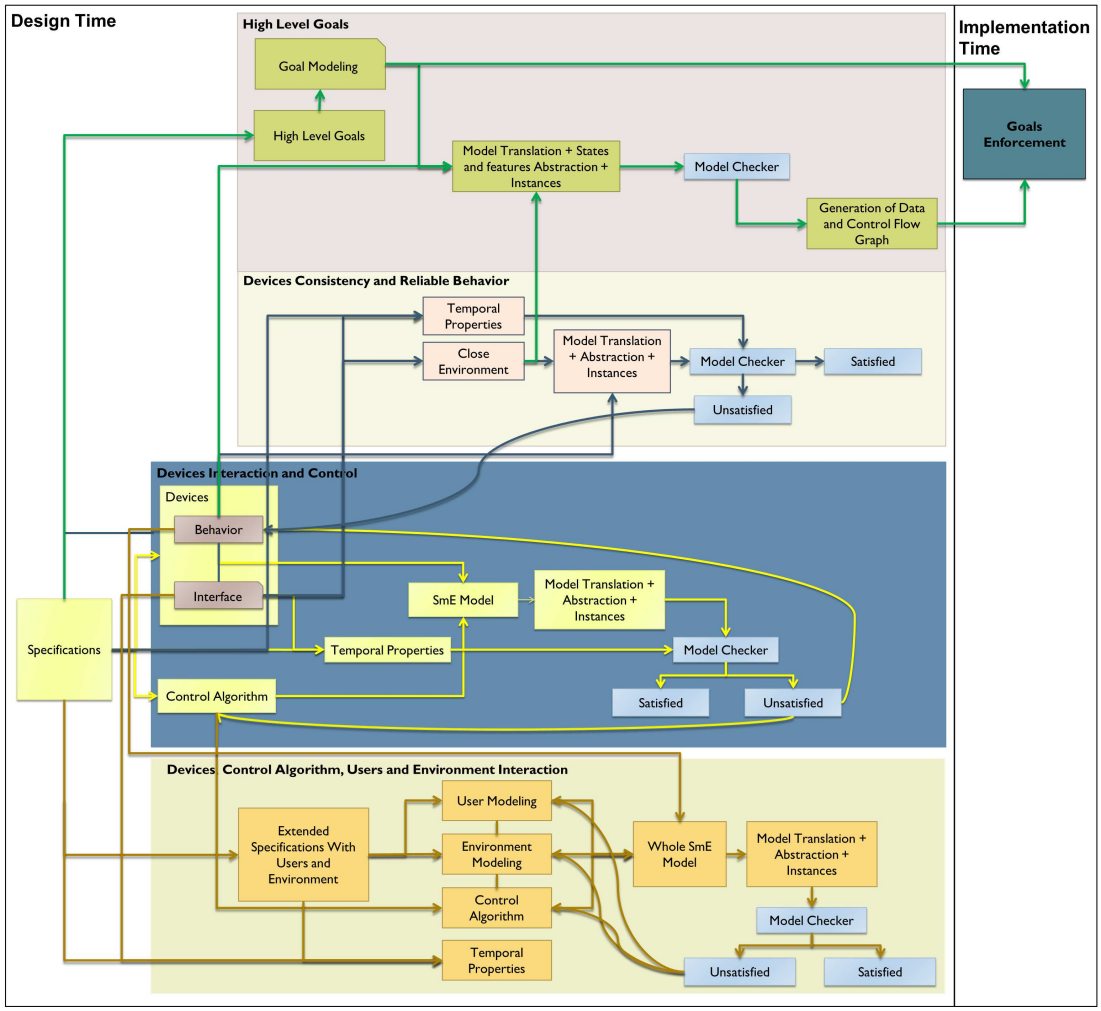

Figure 1.3. Overall Contribution

of the proposed guidelines.

A collective view of the set of technologies used for achieving the desired objective is presented in Figure 1.3. The integration linkage flow, among each section (technique) of the figure, shows the consistency and reusability of verified components. The abstract level listing of the major steps carried out in this thesis is enlisted below:

1. an analysis of the existing state-of-the-art literature, related to the formal modelling and verification of SmE, is performed by proposing a parameter-based empirical methodology

2. lower level modeling of the acceptable actions and behaviors of the users

3. lower level modeling of the devices according to their controlled and offered functionalities
4. lower level modeling of the environment by considering the configurations of the devices and users
5. lower level modeling of the intelligence strategy by which the interaction can be controlled according to the imposed constraints and specified requirements
6. consistency verification among lower level formalisms, which may be adopted for the modeling of components
7. verification of the requirement-accomplished reliable behavior at individual level of the modeled component
8. verification of the reliable interactions among each component
9. verification of the overall imposed constraints of the system
10. design and development of a strategy by which the higher level goals can be achieved

1.4 Structure of the Thesis

The thesis is organized in seven chapters. Chapter 2 presents an overview of the adopted tools with their selection motivation. Chapter 3 presents a parameter-based empirical methodology which is used to analyze the surveyed literature for knowing the trends, covered and uncovered areas by the SmE research community. Chapter 4 presents the proposed methodology used for the formal modeling and verification of SmE. Chapter 5 presents a set of techniques which are designed to implement the proposed methodology. Moreover in this chapter, the reliance of the proposed methodology is confirmed with results obtained by running the case studies (or examples) on these designed techniques. Chapter 6 proposes a methodology for the achievement of high-level SmE user requirements, referred as goals. Chapter 7 concludes the thesis and offers possible directions in which the research can be extended. At the end, in Chapter 8 the list of publications produced from the research explained in this thesis is presented.

Chapter 2

Background

The implementation of the proposed methodology builds upon the existing tools and techniques. The state-of-the art along with the motivation of the selected tools and techniques are explained in the following sub-sections.

2.1 Formal Modeling

Formal modeling is a way of describing the specification of the system by using the syntax and semantics of the mathematically based language. It can be performed with the use of black box and white box modeling conventions. As SmE are huge systems and consist of a number of heterogeneous devices, therefor a generic dictionary and naming/communication convention mechanism may be required which works as a core reference point, and the communication among various heterogeneous devices can be performed by using them. In our case, the generic (centralized) modeling of all the involved devices is performed by using black box technique and the modeling of their inner details is performed by using white box technique.

2.1.1 Black Box Modeling

Controlling and commanding the functionalities of electrical (low cost or smart) devices is one of the main goals of SmE. These devices are of heterogeneous nature, having some common and distinguish functionalities, commands, notification, states and others. The desired functionality from the relevant devices is accessible by posting specific commands, which is acceptable by them. The modeling of these may be achieved by using the object-oriented approaches [19,20], Unified Modeling Language (UML) artifacts [21,22], ontologies or taxonomies [23–29], etc. Ontologies are one of the semantic web artifacts; they provide a formal explicit modeling structure for representing different concepts, their

relationships and their associations, and give a suitable reasoning power on such shared and featured environments [30, 31].

Different ontological solutions exist for the modeling of SmE, such as EHS[1], DomoML [29], SOUPA [27], CoBra [28], DogOnt [23] and some other, each with their own limitations. The EHS taxonomy classifies the Home appliances into white and brown goods along with their placement in SmE. It provides the interaction information so that all the devices can communicate with each other, but it does not provide the information of capabilities, functionalities and desired operations of devices. DomoML ontology uses the existing DomoML-core, DomoML-env and DomoML-fun ontologies. The DomoML-core ontology is used for correlating the components of SmE (described in DomoML-env ontology) and devices along their functionalities (described in DomoML-fun Ontology). It uses different well know vocabularies for defining the concepts. The limitation of DomoML is the lack of state modeling and query functionalities.

SOUPA and CoBra ontologies describe SmE but their main focus is on modeling the functionalities and capabilities of user/agent in the light of pervasive and ubiquitous computing concepts. Their major limitations are the modeling of device functionalities and commands, which are not permitted in SOUPA and CoBra ontologies. DogOnt [23] is an ontology based solution, which is used for the modeling of SmE systems, especially for the modeling of household appliances with their functionalities, commands, notifications, states and placement in the environment. Moreover, for designing a complete context-aware SmE, SOUPA and CoBra ontologies can be used on the upper level from DogOnt: the device interoperability information can be collected with the use of DogOnt, and the context and user/agent information are collected with the use of SOUPA and CoBra ontologies.

DogOnt

DogOnt is an ontology (a semantic web artifact) for the modeling of SmE with a focus on the black box modeling of heterogeneous devices, their relationship with the other devices and their installed location in the SmE [23]. For the modeling of SmE by targeting these goals, DogOnt defines the following top level classifications which are graphically represented in Figure 2.1.

1. “Building Environment:” In this category, the physical environment of the SmE is modeled according to its description, such as building, flat, garage, garden, room (e.g. bathroom, bedroom and kitchen).

2. “Building Thing:” This category is further divided into two main classes: controllable and uncontrollable. In the controllable classification, those devices are

[1]The European Home System, http://www.ehsa.com

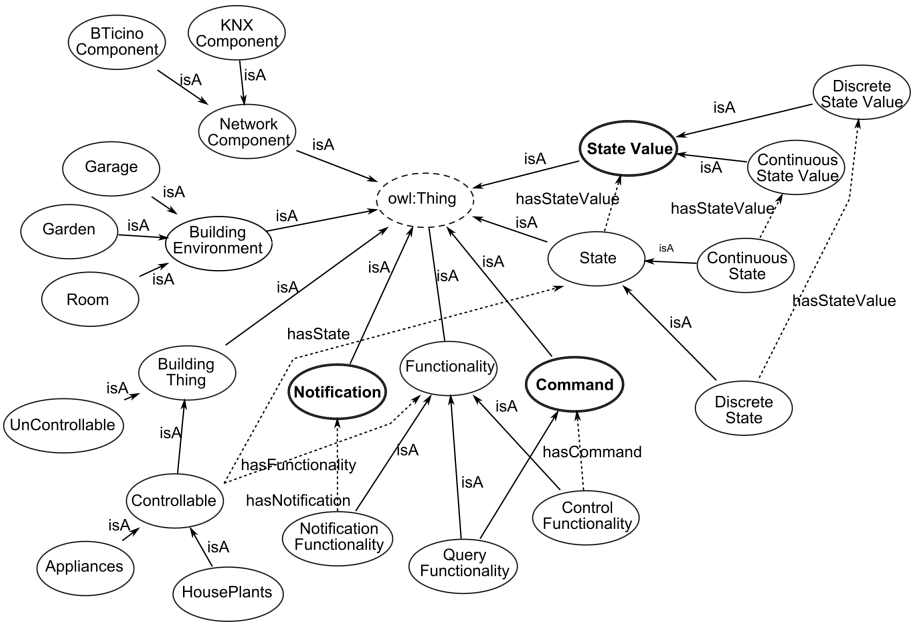

Figure 2.1. DogOnt Ontology

included which can be electronically controlled (e.g. boiler, cooker, fan, lamp, actuator and sensor), whereas in the uncontrollable category are the devices which are part of the SmE but cannot be electronically controlled (e.g. table, sofa and bed). Moreover, for electronically controlling these devices, other electrical devices (e.g. door actuator or window actuator) are required.

3. "Functionality:" The controllable devices are capable of performing some specific functionality. These functionalities can be related to their control (the main functionalities of the devices which they can perform, e.g. a lamp can be *ON* or *OFF*), query (at which state the device is at a particular time, e.g. a lamp is in *ON* state) or notification (the messages/signals which the device sends back after performing an operation, e.g. *onNotification* when the lamp is switched *ON*)[2].

4. "Command:" In this category are the commands which are required for triggering the control functionalities of the devices.

5. "Notification:" In this category, those notification messages/signals are included which devices send after the completion of task. In our case, the system is designed

[2]It is possible that the devices are not so sophisticated, but with the help of relays, hardware (e.g. Bitichno, Knx and ZeeBee) and software (e.g. Dog) gateways, these functionalities can be achieved.

in such a manner that devices send some notification back after the completion of each transition.

6. "State" and "State Values:" At any time the devices are in a particular state, such as a door actuator can be in *moving* or *not moving* state, and the TV can be in *ON* state with the volume *30*. These *ON*, *moving* and *not moving* are discrete state values, whereas the *30* is the continuous state value of the *volume* state. The discrete values are referred as *states* and continuous values as *feature values* during the behavioral modeling of the device. The functionalities associated with these continuous state values are termed as *features*.

7. "Domotic Network Component:" The communication among the devices or gateway (details are given in Section 2.3) is performed through the exchange of messages, which can be carried out by adopting some specific protocols (such as Konnex, ModBus, ZigBee, ZWave and others). For performing actions on the devices (without interacting with multi-interface provided by different manufactures), and bypassing the user exposure to device complexity and variety, the gateway needs to know about the manufacturer and its protocol information so that the devices can understand the messages and act accordingly. The modeling of the information related to device manufacturer, its protocol characteristic and network addressing scheme is performed in this category.

2.1.2 White Box Modeling

Black box information is important and useful for the applications/services, which are required to interact with it. On the other hand, white box or behavioral modeling is required for analyzing the internal behaviour of the device model. Behavior modeling can be performed by using the semantics of labeled-transition systems, in which the more commonly used approaches are UML (a graphical modeling language in Software Engineering) Statecharts, process algebra (like Calculus of Sequential Processes (CCP), Calculus of Communicating Systems (CCS), Algebra of Communicating Processes (ACP)), Petri nets. Each of these can be applicable for the modeling (and verification) of different domains.

Statecharts

Devices range from simple (lamp) to complex (TV) due to their offered features (described in Section 2.1.1 under "State" and "State Values"). These features can be modeled with the use of variables. For controlling them, values of the relevant variables are required to change. For the behavioral modeling of such devices, a semantic is required by which the modeling of multiple features can be achieved. Usually, the modeling of

complex devices is performed with the use of hierarchical and parallel (or concurrent) sub-state concepts.

In 1987, Harel introduced statechart diagrams for the modeling of reactive systems, whose variant became a standard in the UML [32]. Statechart diagrams are used to represent the dynamic behavior, with the support of variables, guard conditions, hierarchical and parallel states of the complex system. The behavioral modeling of devices is performed with the use of statecharts, and the semantics is given below.

Let $\mathcal{D}$ be the set of installed *controllable devices* in SmE. Each device d, $d \in \mathcal{D}$, is characterized by a *device type* (e.g. lamp, television or air conditioner). The behavioral modeling $\mathcal{M}$ of each device d can be defined by a tuple $\mathcal{M}(d) = \{\mathcal{S}, s_0, \mathcal{C}, \mathcal{N}, \mathcal{V}, \Theta, \mathcal{T}\}$ In which

1. $\mathcal{S}$ is a finite set non-empty of states s, that may be simple or composite states.

2. $s_0 \subset \mathcal{S}$ is a finite set of initial states, including the initial states of the composite states.

3. $\mathcal{C}$ is a finite set of triggered commands.

4. $\mathcal{N}$ is a finite set of notifications which a device can produce as an acknowledgment about the status of the assigned task, or an indication message in the case of sensing data by the sensors.

5. $\mathcal{V}$ is a finite set of variables, each variable v, $v \in \mathcal{V}$, is used for controlling the functionality of a device.

6. Θ is an interpretation over $\mathcal{V}$, which is used to assign the values to a variable defined in $\mathcal{V}$.

7. $\mathcal{T}$ is a finite set of transitions. Each transition t, $t \in \mathcal{T}$, is a tuple (s, c, g, a, s') and can be represented as $s \xrightarrow{\{c[g]/a\}} s'$, where s and $s' \in \mathcal{S}$ and s' is the next state to s, $c \in \mathcal{C}$, g is a guard condition over the variables $v_i \in \mathcal{V}$ and a is an action which may consist of (zero or more) notifications, the assignment statements for some variables ($\Theta(v)$, $v : v \in \mathcal{V}$) or both. Moreover c, g and a of the transition tuple are optional and the variable v affected by the transition t will be considered as v' in state s'.

Several transitions may be required to evolve from a source state, ss, to a destination state, ds; $ss \xrightarrow{\{c_i[g_i]/a_i\}} s' \xrightarrow{\{c_{i+1}[g_{i+1}]/a_{i+1}\}} s''$ ... $\xrightarrow{\{c_n[g_n]/a_n\}} ds$. The sequence of these transitions, from a source state to a destination state, is known as an *evolution* e.

2.1.3 High-Level Goals Modeling

A fundamental aim of SmE is to provide intelligent services through which the user can be supported. For the modeling of such services, several approaches are proposed that include device centric perspectives [33–35], learning models [8,36] and abstract modeling frameworks [35, 37–39].

One such approach is the Domotic Effects framework [38, 40] which provides the facility for the modeling of interesting goals (or desires) at high-level as *Domotic Effects* (DE). The framework is organized in a three tiered architecture: Core Layer, AmI Layer and Instance Layer. The core layer has the definitions of basic structure (classes with their relations) by which the services' goals (or DE) can be expressed. The AmI layer assists the SmE designers for defining functional properties in terms of operators. The instance layer has the detailed high-level description of the goals (or DE) based on the particular devices with their desired destination states.

DogEffects

For providing a knowledge-base corresponding to the logical design of DE framework, a three tiered "DogEffects" ontology is formalized [40] based on the OWL Web Ontology Language [41]. By taking the advantages of the modular concepts of OWL (which allows to integrate the ontology with others), the DogEffects is integrated with DogOnt [23] for accessing the required devices (instances in the environment) and their states. A collective graphical view of DogEffects with an example (explained in this following section) is presented in Fig. 2.2. The brief description of each layer is given below.

The core layer defines the three basic concepts of DE framework: Effect, Effect Operator and Operand. The effect (DE) that depends upon a single device (with a specific destination state or sub-state) is called a *simple effect* (SE) and this will be the terminal point of the goal which ends upon the specific device instance name with a particular state value. The DE which depends upon a combination of devices is called a *complex effect* (CE). A CE consists of the functional expressions of SEs or other CEs by using effect operators. These operators can take one DE (called unary operator) or more DEs (called non-unary operator) as operands. Sometimes, the order of operand become extremely important for producing the results and sometimes it is of no value; mathematically, categorized as non-commutative operator and commutative operator, respectively.

The AmI layer contains the operators by sub-classing the general operator classes defined at the core layer. The results of these operators belong to Boolean domain (true or false). Although Boolean logic has a smaller set of operators (e.g. or, and or not), designers are allowed to define their own operators (e.g. alternate, conditional or greater). Against each newly defined operator, the designer is required to provide the implementation details in-terms of Boolean sub-expression. These details are important for encoding

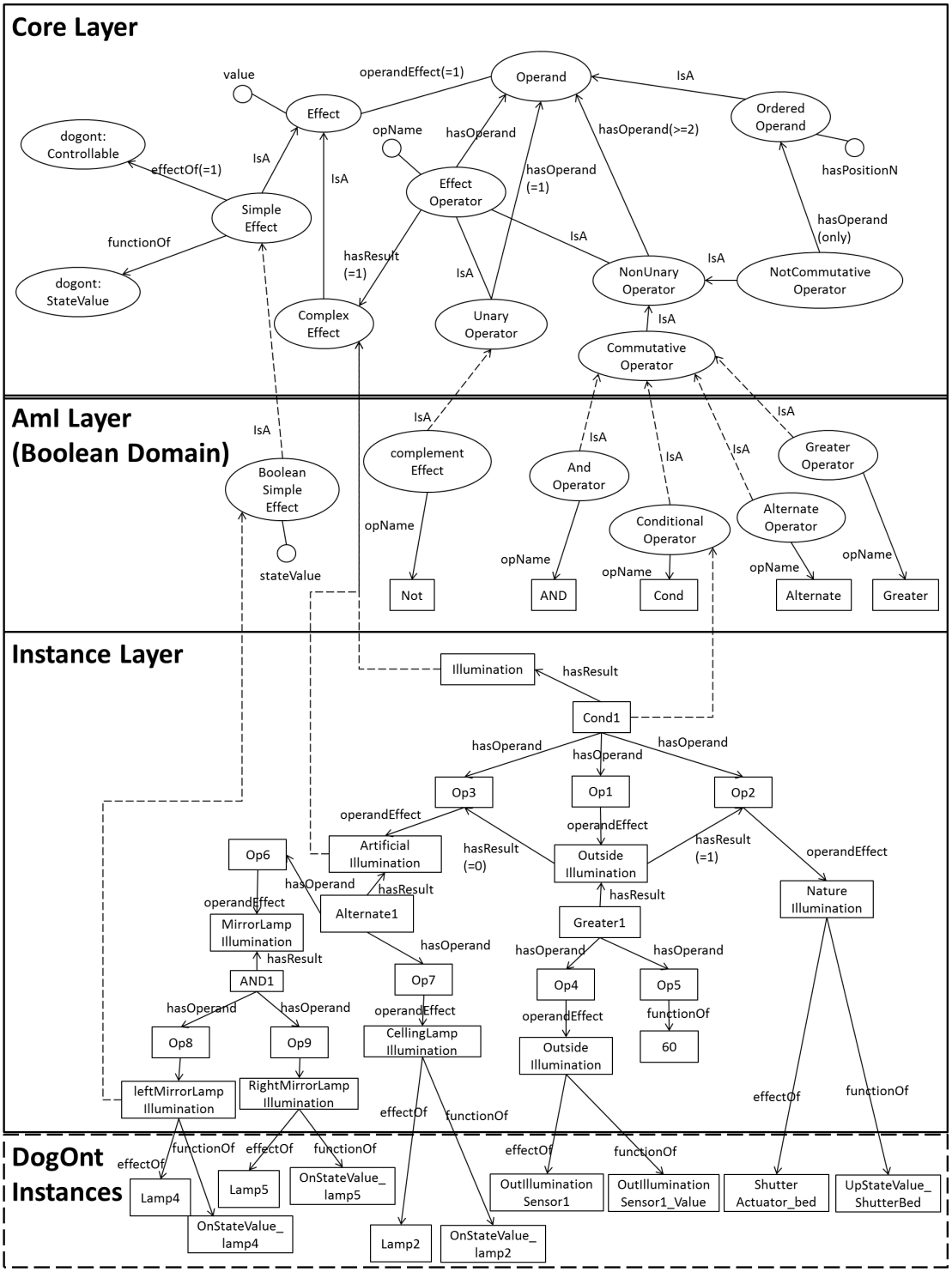

Figure 2.2. An Example of defining Effects

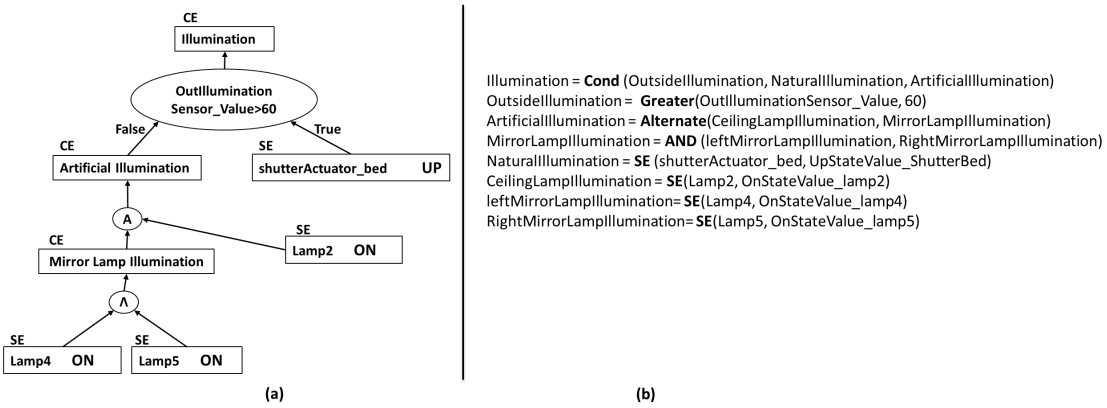

Figure 2.3. A use case for illuminating the Room

in java language and placed in the effect-operator-store with the implementation algorithms of other operators.

The instance layer models the goals as the instances of the classes defined in the core and AmI layers. The following example better clarifies the aforementioned concepts. Consider a room "Illumination" goal, which is graphically represented in Fig. 2.3(a) and functionally represented in Fig. 2.3(b). Depending upon the outside illumination value, the room can be naturally illuminated (by moving the window shutter UP) or artificially (by switching ON the ceiling lamp or the lamps placed at each side of the mirror). This example is modeled at instance layer (instances of the SE, CE and Operators) of DogEffects, in Fig. 2.2, with the association to the device and state instances of the DogOnt.

2.1.4 Properties Modeling

A process, in which the specifications of the system are defined in the form of properties or axioms, by adopting the syntax of some formal language, can be known as formal properties modeling. These properties or axioms are designed to represent behavioral and non-behavioral aspects of the system. The behavioral aspects are related to reliable functionalities (relation between event and action) and non-behavioral aspects are related to security and safety policies, performance, and other characteristics of the system. Formal representation of the concerning specifications are often expressed with the use of temporal logic.

Action-and-State based Temporal Logic

Temporal Logics are used for formally defining the specifications, and based on the set of rules for reasoning with different propositional quantifiers (explicit, implicit) depending on time conditions (Next, Future, Global, etc) [42]. There are two main classifications

of temporal logic: Linear Time Temporal Logic (LTL) and Branching Time Temporal Logic (BTTL). LTL deals with a single trace path at a time, whereas the BTTL deals with multiple trace paths at a time, therefore BTTL is appropriate for the analyzing the complex behaviour of such systems.

UCTL is a UML-oriented action-and-state based branching time temporal logic [43]. It has a combined power of ACTL (Action Based Branching Time Logic) [44] and CTL (State Based Branching time logic) [45]. Due to the rich set of state propositions and action expressions, UCTL is suitable for analyzing the behaviour the system which is modeled in the form of state machines [46, 47]. With the help of UCTL, we can analyze different behaviour of the system like liveness (something good will eventually happen) or safety (nothing bad can happen) with or without the fairness restrictions. UCTL uses the box [ ] ("necessarily") and diamond <> ("possible") operators from Hennessy-Milner Logic [48] and temporal operators (Until, Next, Future, Globally, All, Exists) from CTL/ACTL. By using these logics, the Absence, Existence, Universality and Responses patterns of any predicate(s) can be analyzed.

2.2 Model Checker for UML Statecharts

Model checking [49] is a technique used for automatically analyzing/verifying the behavior of the system, which is dynamic in its nature, according to the modeling. It is capable of exhaustively considering all the states and the possible paths of the model from a particular state for analyzing/verifying the correctness of specifications. State explosion is a major drawback in Model Checking technique; it can occur in complex systems with a large number of states. Abstractions play a vital role for avoiding this issue by preventing unnecessary information of states, variable values and messages from the original model of the system [50, 51]. In-result, the original model is sufficiently reduced in abstract model (the subset of relevant information, which designer want to observe) of the system, which can be conveniently handled by a model checker.

2.2.1 UMC Model Checker

UMC [52, 53] is an "on-the-fly" model checker tool, designed for the formal analysis/verification of the dynamic behavior of UML statecharts, by providing a user friendly environment, for expressing the system and the properties. UMC is fast because it is based on a linear time complexity model checking algorithm for the exact analysis/verification of the system. Moreover, "on-the-fly" nature makes it efficient by not requiring to explore the whole model, but it allows optimally exploring the model based on the given property and return true or false depending upon the satisfaction of the property.

The statecharts semantic in UMC is defined in terms of Double Labelled Transition System (L2TL), which can represent various system configurations on states and system

evaluation through edges [46, 47, 54]. An online version of the UMC model checker is also available[3].

The structure of the model analyzed by UMC consists of classes, instances and abstraction rules. Classes are used to represent the state machines in textual format. They have states, operators (used for synchronous communication of messages) or signals (used for asynchronous communication of messages), local variables and transitions. Further, transitions are associated with states (source and destination), triggers, guards and actions, and instances are the class objects.

2.3 Gateway

In SmE, devices are connected through a (wired or wireless) network with some distributed or central gateways. Dog (Domotic OSGi Gateway) [55] is one such gateway, based on OSGi (Open Source Gateway initiative) [56, 57] framework. It provides a neutral generic interface (API) to the users for performing queries/actions on all the devices (without distinguishing the manufacturers). Moreover, it has additional computation capabilities for making itself technology independent, with the use of DogOnt information of the devices, by accessing the devices with their proper protocol and the suitable addressing mechanism.

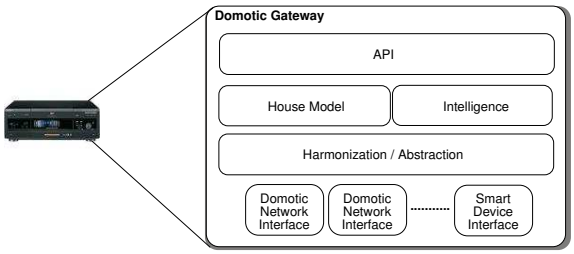

Figure 2.4. Domotic OSGi Gateway

As represented in Figure 2.4, “House Model” contains the Black-Box modeling of the Domotic system; in our case DogOnt provides all the vital information at this stage. The Control Algorithms, which are responsible for the intelligent working of the domotic system, are also embedded in gateways, and represented as the “Intelligence” component. The “Abstraction/Harmonization” layer hides the unnecessary detailed White-Box information of devices/drivers (see Section 2.2.1 for more details). All the instances of the devices, with their behavioral modeling and drivers, are represented at the bottom layer. Dog has additional computation capabilities for making itself technology independent,

[3] http://fmt.isti.cnr.it/umc/

with the use of DogOnt information of the devices, by accessing the devices with their proper protocol and the suitable addressing mechanism.

Chapter 3

Survey and Analysis of State of the art

Due to a large variety of implementation scenarios and support for conditional behavior/processing, the concept of SmE is applicable to diverse areas which calls for focused research. As a result, a number of modeling and verification techniques have been made available for designers. This chapter explores and puts into perspective the modeling and verification techniques based on an extended literature survey.

The formal verification of all possible aspects of SmE is arduous and laborious undertaking owing to the complex nature of these systems. Therefore, research seems to have emphasized the specific aspects based on requirements and their specialized competencies. As a result, this focused approach has deprived the academicians and new researchers/designers from a generic and one-size-fits-all kind of modeling and verification technique. In an attempt to collect the existing state-of-the-art, this chapter brings together the techniques/approaches that are exploited in formal verification of SmE with respect to different aspects with a few overlapping scenarios (such as user interaction, devices interaction and control, context awareness, etc.). The techniques are categorized on the basis of various factors and formalisms considered for the modeling and verification. For this a parameter-based empirical methodology is proposed, which helps to understand the verity of adopted modeling and verification techniques in different applications and scenarios. The study expands upon the uncovered modeling and verification areas of SmE. The findings of the research show that no surveyed technique maintains a holistic perspective; each technique is used for the modeling and verification of specific SmE aspects. The results further help the designers select appropriate modeling and verification techniques under given requirements and stress for more R&D effort into SmE modeling and verification research.

The rest of the chapter is organized as following: the existing formal modeling and verification process adopted for SmE are reported in Section 3.1; the surveyed literature is presented in section 3.2; the proposed parameter-based empirical methodology is described in section 3.3 with the overview of existing state-of-the art; and finally the analysis and concluding remarks, on the surveyed literature against the proposed methodology, are

presented in section 3.4.

3.1 Modeling and verification processes

This section describes commonly adopted modeling and verification processes during formal verification. These processes are classified according to their coverage of SmE aspects and application domain, namely: 1) formal modeling, 2) component modeling, 3) formal verification, 4) Adopted Procedures/Tools. The details of each process are described in the following subsections.

3.1.1 Formal Modeling

Formal Modeling is the process of describing a system (a set of interconnected components performing desired operations) in a well defined formal syntax and semantics language; the following are its different perspectives adopted in the modeling of SmE.

Black Box Modeling

Black Box or Interface modeling is the representation of the information required to interact with the system. The black box modeling focuses on functionalities of the system without any internal details.

Let consider a smart home where whenever a user enters in the bedroom, it will be illuminated depending upon the outside light intensity (user goal– as depicted from Figure 1.1). The smart room senses through sensors the presence/entrance of the user (action). The sensor will send the notification message to the control algorithm. The control algorithm sends a request to the illumination sensor (placed outside the room) that replies with the outside light intensity value. According to this value, the current configuration of window-shutter and the lamp, a control algorithm decides how to illuminate the room (decision): either by moving the window-shutter up or by switching the lamp on. Based on the optimal decision, the control algorithm sends suitable commands to the corresponding devices, which perform the task (operation).

In this, the control algorithm sends a command to the window shutter to move up; the command is fulfilled by the window shutter-actuator. The details about how the command has been sent by control algorithm and how the operation is performed by the shutter-actuator are not considered in the black box models. Instead, the modeling of which message is sent and which action is performed against it are the main focus of the black box.

White Box Modeling

White Box or Behavioral modeling is a representation of a complete internal behavior of the system. The details of how commands are issued, how operations are carried out, and how the system (or individual component) requirements are fulfilled are taken care of in white box modeling. In the running example, for instance, details about how the control algorithm sends commands, how other devices perform their tasks: in other words, complete flow of actions done by the system (or components) is modeled in this category.

Intelligence Modeling

One of the basic objectives of SmE is to provide services in an intelligent way according to the system-level specifications and constraints. To enrich SmE with intelligence, different artificial intelligence and database techniques can be adopted, and their representation is called intelligent modeling. For example, the decision logic of control algorithms either to move window shutter up or to switch lamp off can be modeled by adopting different techniques, such as fuzzy logic, decision trees, rules based, event-condition-action, etc.

Requirements Modeling

Requirements are the starting point of any formal verification process and are specified in the form of properties or axioms by adopting the syntax of some formal language. These properties or axioms are designed to represent behavioral and non-behavioral aspects of the system. The behavioral aspects are related to reliable functionalities (relation between event and action) and non-behavioral aspects are related to security and safety policies, performance, and other characteristics of the system. For instance, in the running example, the requirements related to the events – when the outside light intensity is high then the smart home has to move the window shutter up as well as switch off the lamp (if it is found on) – are presented in formal way in this modeling approach. Formal representation of the requirements is often expressed in temporal logic.

3.1.2 Component Modeling

The components of SmE are users, context/environment, devices and control algorithms. Depending upon the application domain, covered features and the interested scenarios, the (black-box or/and) white-box modeling of these components are accordingly performed. Modeling of these components along with their interaction details are considered in this classification.

3.1.3 Formal Verification

The system correctness with respect to its specifications and constraints can be formally (comprehensively) verified and this process is known as formal verification. During the verification process, different aspects of the system are verified. The description of the noted aspect is presented in the following sub-sections.

Consistency Verification

The consistency verification provides coherency of modeling when both black box and white box processes are applied. It is important to verify that both of the formalisms are consistent with each other; otherwise there is a fair chance that one of the formalisms may have some additional or missing information. Due to inconsistencies, each formalism may behave differently and the access to desired functionality in an independent way may be difficult. For example, if the command to move window shutter up in black box is recognized as “UP”, whilst the same command in white box is identified as “RISE”, this causes inconsistency between the two modeling processes and will lead towards denial of the desired outcomes [58]. Similarly, it is important to verify that the specified requirements are incorporated in the designed model and will behave properly in all scenarios.

Entire SmE Verification

SmE are integrated environments and promise to deliver services in an intelligent requirements-accomplished way. As mentioned in Section 1, SmE covers different aspects of given areas, the verification of the behavior of individual components along with their interaction in the entire system can be formally performed by using model checking or theorem proving techniques, for ensuring the specified SmE behavior, reliable interaction along with the safety and security constraints.

3.1.4 Adopted Procedures/Tools

In this classification, the investigation of the verification processes is performed on the basis of the adopted procedures (through which the comprehensive verification of correctness of system is analyzed with respect to specified requirements) and tools. During the investigation, the maturity of surveyed technique is analyzed in terms of automation, scalability, adopted tool and the examined scenario (case study).

3.2 Surveyed Literature

Various techniques regarding the modeling and verification of SmE and their related components are analyzed under the empirically derived parameters (explained in section 3.3).

Although various literature on SmE is available, the papers considered for this survey encapsulate the formal modeling and verification techniques; providing the SmE developers and designers with a specific study material aimed at collecting SmE-centric work. Though during survey a number of techniques were found which extend the logic for developing the tool ([59–62]), verification of the protocols ([63–65]) and modeling of the cognition-based user behavior ([66, 67]), such techniques are out of the scope of this chapter and therefore are not included.

In [68], the authors present static verification of security requirements for CSCW (Computer Supported Cooperative Work) systems using finite state techniques (model checking). They use a role based collaboration model for specifying coordination and security constraints of CSCW systems. The completeness and consistency of the specification is ensured by verification within the global requirements. They have developed a number of verification models for checking security properties (task-flow constraints, information flow or confidentiality) and assignment of administrative privileges. Their primary contribution is a methodology for verification of security requirements during designing of collaboration systems. Finally, they have run a rather peculiar case study of collaborative activities of academic nature. It is our understanding that replacing the components of this case study with SmE devices, it can also be applied on a complex system.

In [69], the authors propose MIRIE (Methodology for Improving the Reliability of Intelligent Environment) by focusing and motivating on the use of formal methods for the modeling and verification of the reliable behavior of the systems at early design states. The focused components are Users, Devices (sensors, actuators), Control unit and Environment (context) which are attempted to be modeled. The behavior modeling of the system is performed with the use of Promela (Process Meta Language); a language through which the synchronous and asynchronous communication among the components can be modeled as non-deterministic automata and the resultant model can be verified with the use of SPIN model checker. System requirements are specified with the use of LTL temporal logic. Iteratively extending the system model, they explained/guided different properties/features of SPIN model checker. The feasibility of proposed MIRIE is ensured on the Nocturnal (Night Optimized Care Technology for UseRs Needing Assisted Livestyles) project.

In the first part of [10], the authors describe important characteristics, parent technologies and the applications of SmE in various domains. Then they present behavior models of various components of SmE. The modeling of each component is performed by using the semantics of finite state machines (network of automata). The controlling component, known as coordinator system, detects the presence of home occupant with the use of seven motion sensors, placed in kitchen, living room, bedroom and bathroom. By sensing some activity, the system specified constraints are checked and the suitable operations are performed. Also the TV component is modeled: the controller deactivates

the TV when it is found unattended for a long time. Similarly, an alarm manager component is modeled which continuously monitors the triggers from smoke alarms, burglar alarm and emergency pull cord, and contacts fire brigade, security or nursing unit based on triggers. Other controlling components, such as door-bell manager, telephone manager, temperature system manager, environment manager and vital signs monitoring are modeled in the form of state machines. After modeling these components, they design different behavioral properties regarding the verification of specifications-accomplished behavior, individual component behavior, safety and security with the use of Timed Computation Tree Logic (TCTL) by considering the timing factor (real-time system). For verifying these properties on the model, UPPAL is suggested as model checking tool. The process of system modeling and properties designing is performed manually.

In [70], the authors, having worked in the area of verifying AAL (Ambient Assisted Living) systems, present a verification approach for checking the satisfaction of non-functional requirements, such as timeliness and safety based on timed traces semantics and UML-RT models (MEDISTAM-RT). They use a real-time system design and analysis methodology based on the semi-formal UML-RT models (which are generally recognized to be well suited for designing complex time-constrained systems) and the formal CSP+T notation. In their methodology, the system is designed in a stepwise refinement manner, where components are divided hierarchically into sub-components till the final level. The behavior of these basic components are separately designed by Timed State Diagrams (TSD) and the behavior of the whole is derived from the behavior of its constituent parts by following a compositional specification process based on CSP+T. Their methodology is aimed at ensuring safe deadlock-free communication between components. The authors verify an Emergency Assistance System using this verification approach.

In [13], the authors present a formal verification environment for ensuring the desired behavior along with the 'safety' and 'liveness' properties. A case study of Computer Based Railway Interlocking system is reported in which all the communication is controlled through a sophisticated control unit. The system behavior is modeled by using the formalism of Calculus of Communicating Systems (CCS). Just Another Concurrency Kit (JACK) is used as a model checking tool and the properties are specified by using ACTL logic. The model is abstracted by using the "Zooming" technique. In case of need for more reduction, the "Testing signal values" and "Static configuration parameters" techniques can be applied on the model. The system modeling and properties designing process is manually performed.

In [21], the authors present their work for the modeling and verification of SmE. In their methodology, the design process is based on the Systems Engineering standards, especially on EIA-632. In the design process, UML2 and SYSML standard diagrams are used. They take the example of energy manager system (known as ERGDOM) for home comfort. The ERGDOM is a self-configuring system, which identifies the users' comfort patterns, habits and the current temperature of the home, and accordingly makes

the environment comfortable by controlling the functionalities of HVAC systems, shutters, air-conditioning and convectors. The specification related to main functionalities (e.g. Measurement of temperature, moisture, luminosity and air quality in each part of the home for home comfort), the roles of each component (users or devices) and their interaction with the system, is formalized by the use of context diagrams. Then with the adoption of use-case diagrams, the use cases of the system for the desired services (goals) by the users and devices are designed. Further, the behavior of each use case with their interaction is formulated by using sequence diagrams. These sequence diagrams are usually detailed due to the controlling aspects and are then summarized into the concise activity diagrams. These activity diagrams, based on the automatic translation of the ERGDOM model and their relation, are converted into Petri-net by adopting HiLes functional formalism. Various temporal properties, related to the structure and dynamic behavior verification of the control model of ERGDOM, are verified by using TINA (a model checking tool for Petri-net formalism).

In [71], the authors propose a method for the verification of the context and situation in pervasive computing environments. As devices are the basic elements of SmE and each device has some capabilities (features), which can be controlled by changing its value. These values can be non-numeric (a lamp can be on or off) or numeric (the light intensity of dimmer lamp can be controlled from 0% to 100%). During the verification of the context/situation of SmE, it is essential to confirm the particular feature values of concerning devices. A context/situation (defined by some experts) may be associated, through the relation of generalization, composition, dependence or contradiction, with other contexts [72]. The modeling of the context is performed with the use of Context Space Theory (CST) which is further formalized in property format by using Situation Algebra Expression. On the basis of the rules defined in [72–74], they designed 3 algorithms by which the context modeling can be converted into the Orthotope-based situation space and situation algebra expressions [71]. These expressions are further checked on the Orthotope-based situation space for identifying the emptiness or counterexample in case of validation. The feasibility of the proposed methodology is confirmed with the example of Smart Office Environment. It is opportune to note that this paper does not refer the context as the user location.

In [15, 75, 76], the authors present their work for the modeling and verification of ambient intelligence applications. The authors' key focus is on the location dependent movement of users (also referred as ambient) by incorporating the concepts of Pervasive and Ubiquitous Computing. They propose a seven step process for the interface, behavioral and constraints modeling of SmE. The Ontology is used for interface modeling where ambient calculus (AC) (a process calculus formalism derived from pi-calculus) is used for behavioral modeling. They theoretically extend AC by borrowing the concepts from different formal modeling techniques for incorporating the real-time constraints and conditional movements which are among the limitations of AC. The properties related to the pervasive and ubiquitous concepts are specified in terms of Ambient Logic (AL) –

having a combined power of propositional logic, first-order-logic, temporal logic, somewhere and everywhere operators– the properties related to explicit real-time constraints are specified in terms of Real-Time Temporal Logic (RTTL) and the properties related to the pre-and-post conditions are specified by using Design-By-Contract (DBC). A case study of a patient monitoring system is modeled according to the above mentioned formalisms. The patient's movements among different rooms, their activities and operations are identified with the use of RFID Tag. For the modeling and verification, they develop a tool, known as Ambient Designer which can visually model the system in the form of AC and AL. It has an additional functionality of translating the model in the acceptable language of NuSVM model checker. The designed model can be verified by implementing the model checking algorithm for AC in the designed tool or by using NuSVM model checker tool. By this, the properties related to the functional correctness, reliability, availability, safety and security of the system can be verified.

In [77], the authors present a branching time model-checking approach for the formal verification of dynamic aspects of complex systems. Authors defined some formal semantics, based on the work of [78] and JACK (model checker), for considering the dynamic aspects of the system (described in the form of Hierarchical Automata). For the verification, authors consider the Statechart modeling of user interaction with TV system. The dynamic behavioral properties are specified in ACTL logic and verified on the model with the help of JACK model checking tool. The syntax and static semantics of Statecharts are formally defined; however their dynamic aspects are informally defined.

In [79], the authors present the UMC model checker tool for the formal verification of the dynamic behavior of complex systems. The systems which can be verified through UMC are required to be specified in the form of UML communicating Statecharts which can interact with others. The system requirements are formalized by using the syntax and semantics of mu-ACTL logic (ACTL logic with the complete power of mu-calculus as well) and verified on the model with the use of UMC. A case study of the system, consisting of two airports, two passengers and an airplane, is considered for showing the satisfactory outcomes of the model checker.

In [80, 81], the authors present an agent based ambient system for the formal modeling and verification of the interaction among multi-agents. For the generic and domain specific behavior modeling of the interaction, they used predicate logic. And for the verification of the specification, they used rule based Temporal Trace Language (TTL) [82], which is specially designed for the formal specification and analysis of dynamic properties, regarding the qualitative and quantitative (in-term of time) interaction aspects of the systems belonging to biological, cognitive and social domains. They have modeled the Medicine Usage Management system, in which patient takes medicines from the intelligent Medicine box (which has the ability of knowing the quantity of the dosage and the time of previously taken medicine). On crossing the threshold values (maximum and minimum quantity of dosage and time), the system notifies with beep and by automatically sending the SMS to the patient. In case of no reply (or response) from the patient,

the system sends a history SMS to relevant doctor. For the modeling of each component (agent), input, internal and output states are considered in predicate logic format (referred as Ontology). A stochastic model of the patient is considered, and interaction of the model system is sent to the LEADSTO [83], which executes and simulates the traces of the system. The TTL properties are also analyzed on the modeled system (by using these traces) through TTL checking tool [82] or by using SMV model checker[1].

In [86], the authors present a theoretical framework for the formal modeling of SmE by concentrating upon the concepts of Pervasive computing. They perform the formal modeling of requirements, assumptions and behaviors of application software with respect to the user (identification, movements, scopes) and the accessible features of the surrounding devices. According to the requirements and assumptions, the abstract interaction modeling of the accessible features of the devices (by the users at some certain time) is performed, which is further analyzed and formally verified. For the behavioral modeling of such system, Event Calculus is used; a formalism for expressing and reasoning the effects of any action [87]. According to the scopes (the direct interaction of the users with the accessible devices) and duration, the requirements along with the implementation of assumption are modeled in the form of axioms (rules). A theorem proving inference approach is used by adopting Discrete Event Calculus Reasoner [88] as a tool for formally satisfying the system requirements. Discrete Event Calculus is for representing the requirements in the properties format. Discrete Event Calculus is converted into the (well-known) SAT problem and inference is made on the model. With the example of Meeting Support System, they justify the feasibility of their proposed framework.

In [19], the authors present their work related to the modeling and verification of the integrated services in the home network system (HNS). They described and modeled the HNS by using the semantic of Object Oriented modeling in which the environment, appliances, properties, methods, states and other relevant information are considered. After that, they presented a descriptive language for the modeling of the HNS. Then the services' reliability of HNS is verified. Authors used a case study of a home system in which air conditioner, inside and outside thermometers, smoke sensor, ventilators and windows are modeled. The appliances and their integrated services are verified with respect to different CTL specified properties by using SMV model checking tool.

In [89], the authors propose the use of formal methods to analyze the pervasive computing systems. They start with proposing a formal modeling framework for covering the main characteristics of pervasive computing systems. They adopt CSP# for modeling and verification as it is rich in the syntax for modeling concurrent system with hierarchies.

[1]Moreover, Jan Treur (one of the authors of [80, 81]) extended the work by covering other aspects of the agent based ambient system, with the collaboration of other researchers [84, 85]. In his work [85], the cognitive analysis is performed through simulation (therefore is not including in our survey). The purpose of mentioning this is that it is the only found work which performed cognitive analysis for the ambient system.

Later, the safety requirements are identified and the specification patterns for safety and liveness properties are provided as they have classified the important requirements into these categories. By doing so, the critical properties against the system model can be verified by using model checking to detect the design flaws at the early design stage. Finally, a case study of a smart health care system for mild dementia patients (AMUPADH) is run to demonstrate the practicality of proposed framework.

In [90–92], the authors' main goal is to verify the (software of) medical devices by using their UIs. They verify the devices by adopting different strategies. They investigate the user's actual behavior in the field and verify it with the prescribed one as mentioned in the user manual [90]. Similarly, they provide a solution to the investigation authorities for verifying as to which specified user-interface requirements are satisfactorily incorporated in the medical device after their implementation [91]. Moreover, they extend their work and investigate the interaction design issues in the implementation by generating the keying sequences (data entry task) and analyzing them with the user-interaction behavior [92]. For the verification purpose, they adopt theorem proving approach. The Prototype Verification System (PVS) is adopted as a theorem proving tool, and the model of the system is designed by using the reverse engineering processes. The designed model is further translated into the acceptable format of PVS, which is based on higher ordered-typed logic and equipped with similar features of various languages (like C++). The requirements which are required to verify are formalized into axioms (according to the template for properties) and then verified on the model. The verification process is performed in [90, 91] by using proof obligations component of PVS and in [92] by using configuration diagrams (a labeled graph of the modeled system/device in which nodes represent configurations and edges represents transition with guard conditions). These configuration diagrams help in generating the test cases for exposing the interaction issues in the model. With the case study of glucose monitoring procedure in oncology ward [90], infusion pump [91] and a layout of medical device [92], they proved the authenticity of their work.

In [93], the authors present their theoretical contribution for the modeling and verification of pervasive computing environment. They consider the software controlling components, devices, users, environment and other physical objects in the environment as ambient which are spatially interrelated with other objects. Along with the movement, an ambient can enter or leave the environment and can be part of other environments. The modeling of these ambient along with their operations and activities are performed with the use of Ambient Calculus. The properties related to verifying the availability of the services at anytime and anywhere, and devices mobility in case of changing their context (entering and existing of ambient in other environment) are performed with the use of ambient logic. A case study (named as Gaia) of university is considered which is equipped with multiple sensors, computers and actuators. Students can enter with their digital devices (mobiles, PDAs, laptop) and can perform various pervasive activities. Different model checking algorithms/tools, such as specified in [94], can be used for the verification of the pervasive properties.

3.3 Empirically-derived Parameter-based Methodology

For the development of SmE, it is evident from the literature to firstly design and verify the system (motivation is given in Section 1). Practically, project manager (along with the team) may have many questions regarding the modeling and verification of the system. As SmE has the capacity to cover various domains with different perspective, different techniques and tools are used – according to their application areas and covered aspects – for the modeling and verification. On the basis of our experiences and surveyed literature, we try to identify and classify the emerging concerns (listed below) into four groups.

- Among the basic components of SmE (mentioned in Section 1), which components are required to be modeled for this specific application area;
- For the modeling of the selected components, which aspects are required to be covered;
- How the modeling of each selected component is performed by considering the level of details necessary to be achieved?
- How the intelligence/computation is modeled by considering the system constraints?
- How the requirements of different perspectives are modeled, for confirming the correct incorporation in the system model?
- How the verification of the different aspects/perspectives of the components or system is performed?
- Which techniques and tools are used for the modeling and verification of the system?
- Which application area is selected for proving the reliability of the proposed approach?
- Which abstraction technique is employed/adopted for reducing the size of the model so that the verification can be easily performed by focusing on the interested perspective?

During the first course of the literature survey, these questions were identified and classified according to criteria mentioned in Section 3.1. A deep analysis of each classification with the internal categorization is carried out in the second round, and termed as *parameter*. In the third round of survey, the existing state-of-the-art against each parameter is identified and analyzed according to its modeling/verification capacity, termed as *parameter values*.

To the best of our knowledge, the existing state-of-the-art of formal modeling and verification processes (as described in Section 3.1), with respect to their level of adoption and application scenarios, may be comprehensively represented in a tabular form, in which each formal parameter is represented by the adopted state-of-the-art (parameter values) against the surveyed literature. The complete procedure of designing tabular form (from extracting parameters to their corresponding values against each surveyed literature) is referred as empirically-derived parameter-based methodology.

In order to perform an in-depth analysis of the surveyed literature, an overview of the existing state-of-the-art techniques has been performed. The details of their application domains, level of adoption, and their corresponding scenarios are presented in the subsequent sections. Moreover, uncovered areas by the existing state-of-the-art processes and commonly used ones are also investigated.

The following subsections are the main classification, against each of which, a table is designed that provides the adopted tools/techniques information against each surveyed literature. The inner subsections of this classification work as parameters of these tables. These inner subsections represent different perspectives which may be adopted during the formal modeling and verification, in the surveyed literature. The values of the parameter represent the formalism (existing state-of-the art) or the adoption of perspectives in the surveyed literature.

3.3.1 Formal Modeling

Black Box Modeling

Different formalisms are used for Black Box modeling such as Structure diagrams (Class diagrams, Object diagrams) [95] and Ontologies [96]. Structure diagrams are Unified Modeling Language (UML) artifacts that model the Object Oriented systems, whereas Ontologies are the semantic web solution for describing the data as complete data model, formal semantics, knowledge discovery and sufficient reasoning power; due to these advantages, Ontologies are often preferred for the modeling of SmE.

Black box, as a parameter in our methodology, is used for representing the explicitly adopted formalism of modeling information. In tabular format, this parameter is either represented with the name of the employed formalism or with a cross mark (✘), indicating that it is not adopted (as represented in Table 3.1).

White Box Modeling

The behavioral modeling of SmE can be performed through UML behavioral diagrams [95], process calculus [97, 98] and petri-nets [21, 99].

UML behavioral diagrams consist of Use Case, Activity, Sequence, Statecharts and other diagrams. Statecharts (Automata or labeled transition systems) are commonly used

Researchers	Black Box Modeling	White Box Modeling	Intelligence Modeling	Requirements Modeling
Ahmed and Tripathi [68]	✘	Role based collaboration model	Role based	LTL
Augusto and Hornos [69]	✘	Activity Modeling Through Promela processes	Event (Activity detection), Condition(location identification), Action (operation graded)	LTL
Augusto and McCullagh [10]	✘	Finite State Machine	Event Condition Action	TCTL
Benghazi et al. [70]	✘	UML-RT (Timed Sequence Diagram, Timed State Diagram), CSP+T	Event Condition (previous history) Action	F_{TT} (Common Formal Semantic Domain)
Bernardeschi et al. [13]	✘	CCS/MEIJE Process Algebra	Event Condition Action	mu-ACTL
Bonhomme et al. [21]	System Engineering Standards, EIA-632, Use Case, Sequence, Activity and Dynamic Context Diagrams, UML2, SYSML	Petri-Nets, HiLes	Decision Logic	Temporal Properties
Boytsov and Zaslavsky [71]	Context Space Theory (CST)	Orthotope-based Situation Space	Weighted Rule Based	Situation Algebra Expression
Coronato and Pietro [15, 75, 76]	Ontology	Ambient Calculus	Ambient movement, Pre-and-Post conditions	Ambient logic + RTTL
Gnesi et al. [77]	✘	Hierarchical Statecharts	Event Condition Action	ACTL
Gnesi and Mazzanti [79]	✘	Communicating State Machines	Event Condition Action	mu-ACTL
Hoogendoorn et al. [80, 81]	✘	Predicate logic	Rule Based	TTL
Ishikawa et al. [86]	✘	Event Calculus	Rule Based	Axioms Based through Discrete Event Calculus
Leelaprute et al. [19]	Object Oriented Modeling, System description	Object Oriented Modeling, Service description	Event Condition Action	CTL
Liu et al. [89]	✘	CSP#	Rule Based	LTL
[90–92]	✘	PVS Logic, a Typed higher-ordered Logic	✘	Axioms Based (according to property template)
Ranganathan and Campbell [93]	✘	Ambient Calculus	Rule Based, DL-Based, Relational Algebra	Ambient Logic

Table 3.1. Modeling Evaluation

artifacts for specifying the system in a formal way. Different variants of state diagrams for modeling different aspects of behaviors, with each variant having its own limitations, are designed. The more famous and exploited variants are Harel Statecharts [32], Communicating Statecharts [79], Automata [10, 100] and Hierarchical Automata [77, 101]. The probabilistic and timed behavior of the complex system can be modeled with the use of Probabilistic Statecharts [102] and Timed Automata [16], respectively.

Process algebras can also be represented as labeled transition systems for specifying the behavior of the system. In process calculus, the most commonly used formalism are Calculus of Communicating Systems (CCS) [13, 103], Communicating Sequential Processes (CSP) [104, 105] and Pi-calculus [106], whereas their extension with the context-aware (mobility) modeling information is known as Ambient Calculus (AC) [15, 107]. The probabilistic modeling of the system is mostly performed by enhancing the semantics of process calculus formalisms.

Petri nets are used as framework for specifying the concurrent systems with detailed (mathematical and conceptual) basic semantic for their modeling. Timed-petri-nets is an extension of petri-nets, in which the concurrent behavior of the system is formally specified in terms of time.

White box, as a parameter in our methodology, is used for representing the adopted formalism of modeling information. In tabular format, the value of this parameter is represented with the name of the employed formalism.

Intelligence Modeling

For providing services intelligently, different techniques are adopted among which artificial intelligence (e.g. fuzzy logics in [108, 109], decision trees in [110], machine learning in [111], case-based reasoning in [112], rule-based reasoning in [71], databases (e.g. event-condition-action in [113] and SQL-based data management in [114]) are some of the mostly adopted approaches. Based on these approaches, control algorithms decide feasible operations and send commands accordingly to corresponding devices.

In empirically-derived parameter-based methodology, intelligence modeling is used as a parameter (see Table 3.1). The value represents the name of the employed formalism by the surveyed technique and cross mark (✘) indicates that it is not observed in the surveyed technique (as represented in Table 3.1).

Requirements Modeling

Temporal Logics are widely used in formal verification in order to formalize and specify the requirements of complex systems [42, 44, 45, 54]. The truth value of these specified requirements depend upon time; whether the specific requirement will be true at any path (Exists), or on all the paths (All) of the complex systems. In addition to Exists and All, there are other temporal quantifiers like Global, Next, Future, Until, Implies, which help

in verifying the complex requirements on different branches from some specific state at a certain time.

Linear-Time Temporal Logic (LTL) is used to represent the requirements for linear time model of the system, whereas Action Based Branching Time Logic (ACTL) [44] and State Based Branching Time Logic (CTL) [45] are used for representing the requirements for computational time temporal logic of the system. Several logics are designed for handling different aspects of requirements, many are formulated by integrating the already designed languages addressing a wider range of requirements like UCTL [115], SocL [116]. Time based requirements are usually handled by TCTL, RTL, RTTL, TPTL, RTCTL [117] whereas probabilistic requirements are handled by using PLTL and PCTL logics [118].

In our methodology, requirements modeling is used as a parameter and the value (in Table 3.1) reports the adopted logic by the surveyed technique.

3.3.2 Component Modeling

User Modeling

Users interact with the SmE in their own ways which, in turn, responds according to the specified and modeled behaviors. The level of details and sophistication varies from system to system, context to context and goals to goals. Among different perspectives, some of the behavioral aspects which are considered for user modeling are:

- User identification (UI): the identification of the user through sensing and/or input devices;
- User actions history (UH): the stored history of previous user actions;
- User privileges –on the basis of their roles– (UPr): based on the role categorization, the system functionality provision granted to the user;
- User position –pre- and post-action execution– (UP): the geographical location of the user within the system boundaries with respect to a specific action;
- User's possible actions (UA): the actions of the user which can be contemplated and facilitated by the system;
- User's possible behaviors (UB): the behavior (related to movement and context-approved actions) of the user which can be contemplated and facilitated by the system;

In Table 3.2, the values at the end of listed items (placed in parenthesis) are used as parameter values for representing the modeling aspects covered by the referring technique.

Researchers	Users Modeling	Devices Modeling	Control Modeling	Context Modeling	Interaction Modeling
Ahmed and Tripathi [68]	UPr, UA		✔		UI, IC, CO
Augusto and Hornos [69]	UI, UP, UA, UB		✔	✔	US, UC, SC, CO
Augusto and McCullagh [10]	UA	Behavior	✔		US, UI, SC, IC, CO
Benghazi et al. [70]	UH, UA		✔		US, UI, SC, IC, CO
Bernardeschi et al. [13]			✔		IC, CO
Bonhomme et al. [21]	UI, UH, UA		✔		US, UI, SC, IC, CO
Boytsov and Zaslavsky [71]			✔		IC
Coronato and Pietro [15, 75, 76]	UI, UP, UB		✔	✔	US, UC, SC, CO
Gnesi et al. [77]	UA	Behavior			UI
Gnesi and Mazzanti [79]	UA, UB	Behavior		✔	UC, UI
Hoogendoorn et al. [80, 81]	UH, UA		✔		UI, IC, CO
Ishikawa et al. [86]	UI, UPr, UP, UA		✔	✔	US, UC, UI, SC, CO
Leelaprute et al. [19]		Behavior	✔		IC, CO
Liu et al. [89]	UI, UA		✔	✔	US, UC, UI, SC, IC, CO
[90–92]	UI, UPr, UA	Behavior			UI
Ranganathan and Campbell [93]	UI, UA		✔	✔	US, UC, UI, SC, IC, CO

Table 3.2. Component Modeling

Devices Modeling

Device modeling can be done by two methodologies: interface and behavior. In interface modeling, we usually consider commands (triggers) a device may receive; associated functionality (operation) it may perform; constraints (rules) it has to follow; states at which it will be at any time; notifications that it sends after the completion of task. Whereas in behavior modeling, acceptance of specific commands on a particular state, implementation of constraints, operations which may be performed on that state after the satisfaction of constraints are considered.

Referring to Table 3.2, the value "Behavior" under this category show the modeling of internal behavior of the devices in the surveyed technique.

Control Algorithms Modeling

The overall sophisticated control strategy of SmE is implemented through control algorithms. Control algorithms take input from the input/sensing devices and according to the system specifications and imposed constraints, decide for the reliable functionality. For the fulfillment of the desired functionality, they send commands to the relevant operating devices for performing required task/operation (as presented in Figure 1.1).

In Table 3.2, a tick mark (✔) under this parameter show that the referred technique takes decision by implementing the mentioned pattern.

Context/Environment Modeling

The identification of the user location is grouped in this category, and termed as Context modeling. Referring to Table 3.2, a tick mark (✔) under this category shows the referring technique performed this type of modeling.

Interaction Modeling

SmE components can interact with each other for the achievement of desired goals. In the surveyed literature, researchers are found focusing on different interaction levels and accordingly building the system. On the basis of these focuses, we categorized the interaction levels into the following groups:

- User interaction with the environment through sensors (US): the considerable user actions in the environment are monitored or noticed with the use of sensors;
- User interaction according to its context (UC): the user actions are recognized according to user's movements in the environment; although these are usually monitored by sensors, the focus point is that with a change in the position, the system will able to consider the activities;

- User action performance on input devices (UI): user interacts with the system through handheld devices, or by directly performing action on the real inputting devices;
- Sensor interaction with the control algorithms (SC): the sensed data is sent by the sensors to the control algorithms, on the basis of which control algorithms decide for the preferable action;
- Input device interaction with the control algorithms (IC): the handheld devices or real devices send the commands to the control algorithms for performing the specific task;
- Control algorithms interaction with the operating devices (CO): control algorithms incorporate the intelligence strategies and on the basis of incoming commands, decide for the preferable action and accordingly send messages to the relevant devices.

In Table 3.2, the values presented at the end of each listing item are used as the parameter values for informing that the referred technique is focusing/performing on which type of interaction modeling.

3.3.3 Formal Verification

The system correctness with respect to its specifications and constraints can formally (comprehensively) be verified and this process is known as formal verification. Different aspects are verified during the verification process. The description of each aspect is presented in the following subsections.

Consistency Verification

Consistency verification, as a parameter in our methodology, is used for representing whether applied modeling formalisms are consistent with respect to their vocabulary and functionalities, and the specified requirements are properly incorporated in the designed system (as mentioned in Section 3.1.3). In Table 3.3, a tick mark (✔) shows it is considered and performed in surveyed literature.

Entire SmE Verification

In order to verify entire system, different aspects are covered, which can be classified as the following: Users Behavior Verification, Context Verification, Device Behavior Verification, Devices Interaction and Control Verification, Real Time Verification and Probabilistic Verification.

Authors	**Consistency Verification**	Entire System Verification					
		Users Behavior Verification	Context Verification	Device Behavior Verification	Devices Interaction Control Verification	Real Time Verification	Probabilistic Verification
Ahmed and Tripathi [68]		✔			✔		
Augusto and Hornos [69]		✔	✔		✔		
Augusto and McCullagh [10]				✔	✔	✔	
Benghazi et al. [70]	✔				✔	✔	
Bernardeschi et al. [13]					✔		
Bonhomme et al. [21]	✔(Behavioral Analysis)				✔		
Boytsov and Zaslavsky [71]	✔						
Coronato and Pietro [15, 75, 76]		✔	✔			✔	
Gnesi et al. [77]				✔			
Gnesi and Mazzanti [79]		✔	✔				
Hoogendoorn et al. [80, 81]					✔	✔	
Ishikawa et al. [86]		✔	✔		✔	✔	
Leelaprute et al. [19]				✔	✔		
Liu et al. [89]			✔		✔		✔
[90–92]	✔			✔			
Ranganathan and Campbell [93]		✔	✔		✔		

Table 3.3. Formal Verification Evaluation

- *Users Behavior Verification:* The key concern while designing SmE is to facilitate the environment with integrated technologies to benefit users, who have certain goals/desires and a complex web of behaviors which can be adopted during interaction with the system. In this classification, accomplishment of user goals against the specified actions with the input devices (or sensors) and the understanding of the possible behavior (moves) of the users are verified. The tick (✔) sign under this category shows its application in verification of users actions and behavior.

- *Context Verification:* Users interact with the SmE through the environment. According to location (also referred as Context), users can access services (mostly concerned with safety and security) from the environment. The environment models of SmE have extra computational power for determining the current state of the corresponding objects/devices/users and providing specified services accordingly. For instance, room illumination services are only accessible when residents are awake and/or present in room. Table 3.3 reports whether the surveyed technique performs context verification or not; the tick (✔) sign shows context verification is performed.

- *Device Behavior Verification:* The devices in SmE are of heterogeneous nature with some common and distinct features. They are self-dependent components with their own internal specified behavior that may be complex based on the device features (smart devices). In this classification, the specified internal behavior of devices is explicitly confirmed on their models. The tick (✔) sign in Table 3.3 shows scenarios in which this verification is performed.

- *Devices Interaction and Control Verification:* The system level requirements are implemented through control algorithms which regulate interaction among devices. In this classification, the system level constraints and the reliable interaction among devices under control algorithms are confirmed. The tick (✔) sign under this category shows it has been applied.

- *Real Time Verification:* The application areas of SmE are almost in every domain. Some applications can be time dependent such as traffic control system, where time factors are also considered in modeling and verification stage. In this classification, real-time verification of the system is ensured. The tick (✔) sign in Table 3.3 indicates real time verification is performed.

- *Probabilistic Verification:* The system being large along with possibilities of its multi-tasking nature make it more complicated. Probabilistic modeling, in this regard, can be adopted to ensure its smooth behavior with respect to possible actions the system can perform at a given time. SmE may encounter challenging conditions such as versatile user behaviors, malfunctioning sensors, broken or out-of-order devices, which may compromise reliable response of the system. To cater to such

scenarios, probabilistic modeling and verification is usually performed. In this classification, checking of probabilistic verification in surveyed literature is taken into consideration (see Table 3.3).

3.3.4 Adopted Procedures/Tools

In this category, analysis of verification procedures/tools is considered. The following subsections explain in details.

Formal Verification Techniques

Model checking is suitable for the system in which the state space is finite [122] but it can also work for infinite state space models represented as a finite state space by adopting some reduction technique (such as abstraction, inactive variable elimination, internal transition by passing, approximation). Several model checking tools are available for the formal verification of SmE related systems. The verification can be performed using Linear-Temporal Logics or Branching-Time Temporal Logics. Some of the reported model checkers that use Linear-Temporal Logics are in [119–121], HEGO [123], vUML [124] based on SPIN [125], whereas JACK, [77], SMV [126], CMC [127] and UMC [79] are used for verifying state and action based branching time temporal behavior. For the verification of real-time systems, UPPAL [128] and nuSVM [129] model checkers are used, while TINA [130], TAPAAL [131], ROMEO [132], DREAM [133] are exploited when model is specified in terms of Petri-Nets. Time based verification can be performed with the use of UPPAL, TAPAAL, ROMEO, DREAM, CWB [134] and other model checkers, whereas probabilistic model checking can be performed with the use of CADP [135], PAT [136], PRISM [137] and others.

The formal verification on the system can also be performed with the use of theorem proving techniques, in which the system is modeled using invariants and set-theoretical structures. Different logical inference rules, linear and temporal properties can be applied for checking correctness of the system. Inference can be semi-automatic (with user involvement) or fully-automatic (by providing full power of inference to theorem prover). The commonly used semi-automatic theorem prover are HOL [138], Coq [139], ACL2 [140] PVS [141] and Isabelle [142], whereas fully-automatic theorem prover are Perfect Developer [143] and Escher C [144]. The possible scenarios, where these modeling techniques are applied, are described in Table 3.4.

Abstraction

In case of model checkers, abstraction techniques are frequently used for reducing the size of the system model. The abstraction can be applied on states, actions and variables of the model. Under this parameter, either the name of the abstraction technique explicitly

Researchers	Verification Technique	Abstraction	Automatic	Scalability	Verification Tool	Case Study
Ahmed and Tripathi [68]	Model Checking	incremental modeling with separation of concerns and property specific abstractions	Automatic		SPIN	Computer Supported Cooperative Work (CSCW) system for Monitoring Exam Activities
Augusto and Hornos [69]	Model Checking	✘	Manual	✔	SPIN	Nocturnal (Night Optimized Care Technology for UseRs Needing Assisted Livestyles)
Augusto and McCullagh [10]	Model Checking	✘	Manually	✔	UPPAL	Smart Home
Benghazi et al. [70]	Transformation and Mapping rules	✘	Semi-automatic	✔	✘	Emergency Assistace System for Cardiac patient
Bernardeschi et al. [13]	Model Checking	Testing Signal Values, Static configuration parameters, Zooming	Manually		JACK	Computer Based Railway Interlocking System
Bonhomme et al. [21]	Model Checking	✘	Semi-automatic		TINA	Smart Energy Management System for Home Comfort (EDGDOM)
Boytsov and Zaslavsky [71]	Rule Based	✘	Manual		Self-designed Algorithms for Emptiness Check	Smart Office Environment
Coronato and Pietro [15, 75, 76]	Model Checking	✘	Semi-automatic	✔	Ambient Designer, Nu-SMV	Pervasive Healthcare Application for Monitoring the Patient
Gnesi et al. [77]	Model Checking	Refinement Function	Manually		JACK	User and TV System
Gnesi and Mazzanti [79]	Model Checking	not generating the global model of the system	Manually	✔	UMC	Plane and Passenger in Airport System
Hoogendoorn et al. [80, 81]	Model Checking	✘	Semi-automatic		TTL Checker, SMV	Medicine Usage Management
Ishikawa et al. [86]	Theorem Proving	✔	Manual		Discrete Event Calculus Reasoner	Meeting Support System
Leelaprute et al. [19]	Model Checking	Symbolic representation of the State space	Semi-automatic		SMV	Air Cleaning Service in Home Network System
Liu et al. [89]	Model Checking	✘	Semi-automatic	✔	PAT	Heath care system for Dementia patient
[90–92]	Theorem Proving	✘	Semi-automatic		PVS	Glucose monitoring procedure in oncology ward, Infusion pump, a real medical device
Ranganathan and Campbell [93]	Model Checking	✘	Manually		specified in [119–121]	Gaia (pervasive environments with digital devices)

Table 3.4. Adopted Procedures/Tools

adopted by the surveyed literature is mentioned or the cross (✘) sign indicating that it is not performed.

Automated

This parameter is used to represent that the surveyed technique generates the model and the properties “automatically”, or it performs some manual instruction and some part is automatically generated (“semi-automatically”), or all work is performed “manually”.

Scalability

Scalability is among the important factors which are considered for the evaluation of techniques. It is a broader term and can be used in many dimensions. Here the scalability is referred as the ability of the surveyed technique to enhance itself by adding more components of same or different nature in the system. Under this parameter, the tick (✔) sign indicates our observation that the technique can be enhanced by adding other components with their inner aspects and details.

Verification Tool

This parameter indicates that among the several model checking/theorem proving tools (as listed in section 3.3.4), which tools are adopted and in which domains and scenarios. In Table 3.4, this parameter (verification tool) contains the name of the applied approaches/processes/tools in verification process.

Example/Case Study

This parameter has the name of the application area, which is selected by the surveyed literature as a case study/example, for proving the satisfactory outcomes.

3.4 Discussion

In this chapter, a survey of SmE modeling techniques is empirically conducted. In survey, the modeling techniques which also perform formal verification for confirming their correct behavior are considered.

As evident in the Table 3.1, the analysis shows that most of the techniques do not perform black box modeling, but white box modeling is globally performed. The reasoning behind this trend can be attributed to the fact that at least behavior modeling is performed in any case due to the minimum formalism requirement. Black box modeling, on the other hand, plays more of a foundational role (in form of common dictionaries and conventions). This role nevertheless has a considerable planning and development cost.

Owing to shortage of time and resources, researchers seem to be in a hurry to furnish the obvious functioning aspects of formal verification rather than the foundation. For white box modeling, most of the techniques consistently use Statecharts (or their variants) due to their maturity and ready availability other than mathematical-oriented wide coverage of all possible paths. Intelligence modeling, mostly provided through Event-Condition-Action technique, is almost globally performed by all the techniques. It is imperative to mention that artificial intelligence (fuzzy logic, decision tree) is not diffused in formal verification practice. Finally, using the variants of temporal logic, most of the surveyed techniques perform requirements modeling.

The analysis of Table 3.2 shows that minimal user modeling is performed by almost all the surveyed techniques. However, it is opportune to mention that most of the techniques acquire the knowledge of user identification and actions to perform user modeling. The real user behavior modeling is performed by a minority of techniques and a majority does not do so due to pertinent complexity of behavior versatility and uncertainty. Further, all surveyed techniques in device modeling are considering the interaction between the devices. But the behavior of individual devices is only modeled by a very few techniques. Further the table shows that almost all the techniques perform control modeling, but their point of reference is different: some involve the user's perspective, some involve device's perspective and some involve the environment's/context's perspective. 6 of the surveyed techniques consider the user movements before taking any decision in context modeling. Further, the interaction modeling seems to be largely covered by the techniques; user identification and action, and based on them the operations which could be performed are major focus of interaction modeling. Leaving aside Liu et al. ([89]), no other technique seems holistic and global in its nature. They consider one or the other component of SmE with the control and model it; mostly the user. Three of the techniques are somewhat holistic as they model 3 out of 4 SmE components. There is definitely scarcity of techniques covering all areas of components modeling.

The analysis of Table 3.3 shows that only 3 techniques perform holistic consistency verification (between black box and white box), whereas Ahmed and Tripathi ([68]), though not having performed a black box modeling, still adopt a consistency verification strategy by validating the successive formalisms with the previous ones. Since most of the techniques have not performed black box modeling, therefore it seems appropriate that they (other than Ahmed and Tripathi) do not perform consistency verification. It is also observed that 6 techniques perform user behavior verification. This shows a lack of interactivity and liveliness of SmE modeling and verification practices.

Similarly, the situation is equally alarming in context verification with 6 out of 16 surveyed techniques performing this verification. It can be argued that SmE are context critical systems and demand an understanding of their physical and location-based modalities, therefore such a modeling is highly required and the research impetus is too strong to ignore in future works.

Also, some of the techniques are found to perform device interaction verification. The

increasing complexity of devices and intricate nature of their behavior within the system impede this type of verification. But, based on mounting needs, it is imperative to perform this type of modeling.

The analysis further reveals that device interaction and control verification is performed by almost all the techniques. It is grounded on the fact that most of the surveyed techniques use control algorithms for accomplishing the system requirements, therefore it seems natural that all these techniques do perform this kind of verification. Finally, real time verification and probabilistic verification are not found so diffused in the surveyed techniques. These seem to be highly neglected areas of SmE verification and owing to their importance, it is necessary that SmE researchers also divert some effort to these areas.

The analysis of Table 3.4 shows abstraction is performed by less than half of the surveyed techniques, whereas others do not perform the abstraction. It can be said that those techniques which perform abstraction do so based on their large size and focus. According to the observation, it is found that some techniques are scalable, which can be enhanced by adding the other components and their aspects in more details. Further, it is found that 8 techniques are manual, 7 are semi-automatic and only 1 technique claims to be fully automatic (as it is rule-based). It can be argued there that the complex nature of SmE and correspondingly complex modeling and verification requirements hinder the automation of these techniques, as the only fully automatic techniques is also not truly automatic in its nature and is based on rules. All but two surveyed techniques use verification tools of different nature. Finally, all the surveyed techniques are tested on one or the other case study of varying nature, scope and level of complexity.

Chapter 4

Proposed Methodology

A comprehensive methodology is proposed for the design and verification of SmE. The methodology entails all the major components of SmE; users, devices, environment and control algorithms. It is advisable that for designing the SmE, the detailed specifications of these components are listed at requirement gathering phase. The organized specifications provide a better understandability of the system (and its related components) through which the ambiguities during modeling can be sufficiently reduced. Further, these organized specifications help in designing the properties related to the verification of reliable behavior (consisting of safety, security and other major aspects) of the system. For the behavioral modeling of each component, the methodology adopts Statecharts. The methodology provides ten steps, which are briefly explained in with the following case study of Bank Door Security Booth System.

4.1 Bank Door Security Booth System (BDSB): A Case Study

The Bank Door Security Booth System (BDSB) is our real world example of a SmE system [113], which is extended with the concept of users' and environment (context) modeling. Although BDSB is an initial level small SmE system, it exhibits a complex behavior due to the interaction of multiple users with the system and performs a complex communication between different hardware (e.g. devices) and software (e.g. control algorithms) components according to user interaction. A graphical layout of the BDSB environment is presented in Figure 4.1.

The BDSB is designed in such a way that multiple users can interact with the system; ideally, the security and safety measures of the BDSB system should never fail. The system is composed of two electronically controlled doors, located outside (known as external door (DExt)) and inside the bank (known as inner door (DInner)). For electronically controlling a door, actuators are installed. DExt and DInner are controlled by DAExt and

DAInner door actuators respectively.

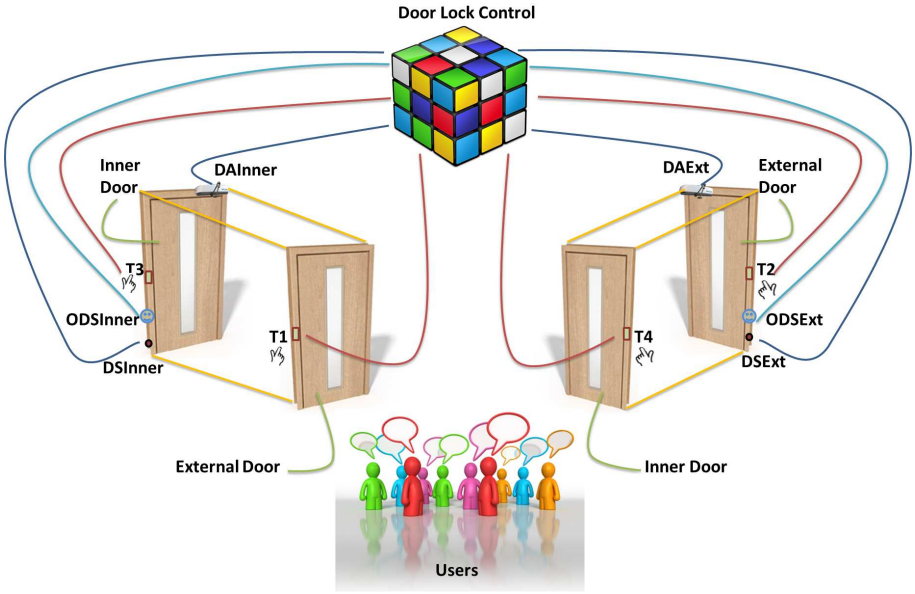

Figure 4.1. Bank Door Security Booth System

There is an isolated space between both doors, where users have to wait so that the opened door is closed first and then the other door may be opened. The user request for door opening is only possible through touch sensors (TS), which are installed near each side of each door. The Touch sensor attached outside the DExt is called T1, and the one attached within the isolated space is called T2. Similarly, the touch sensor attached to the DInner from within the isolated side is called T3 and the one attached from inside of the bank is called T4.

The Door sensors (DSExt and DSInner) are used for querying the status (whether it is open, close or in moving states) of the door; DSExt is attached with DExt and DSInner is attached with DInner. Similarly, two obstacle detection sensors are used for reopening the door when it is in closing state and any object (e.g. person) is held in between the closing path of the door, ODSExt is attached with DExt and OSDInner is attached with DInner. A control algorithm, known as Door Lock Control (DLC), manages all the communication and functionalities of these devices in a safe and secure way.

4.2 Methodology

The explanation of the proposed methodology with the design details is given in the following sections.

4.2.1 Step 1: SmE Specification Identification

Requirements gathering and listing in a suitable way is normally the first step from where any complex project begins [95]. The same process is adopted for the design of SmE where the system level specifications are identified. These are related to the physical components of the system, their functional behavior (along with their interaction details) and the overall constraints (e.g. Security, Safety) for the designed SmE. A graphical view of the activities carried out in this step is explained in Figure 4.2.

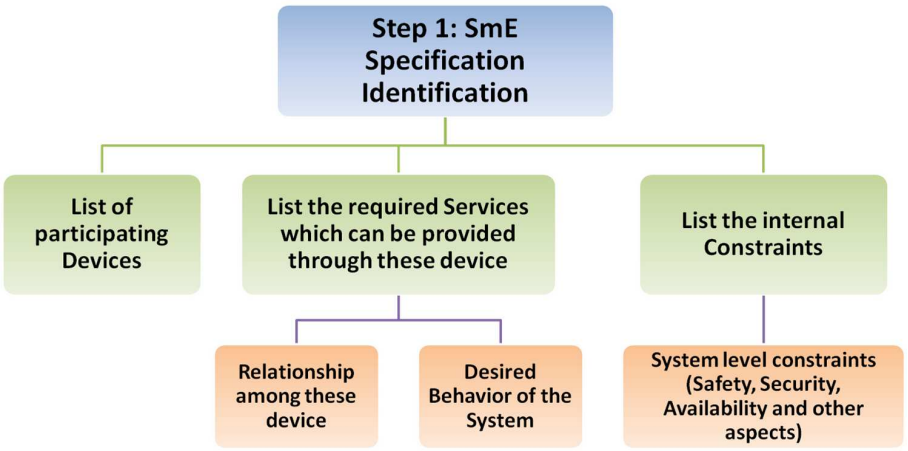

Figure 4.2. SmE Specification Identification

The devices which are used in SmE are of two types: Controllable and Uncontrollable. Controllable devices can be divided into two main categories based on their usage: input and output devices. Input devices are used for taking the input from the environment, by observing the actions of the users (e.g. sensors) or with the direct interaction of the users to the devices (e.g. touch sensors); whereas the output devices (e.g. actuators) are used for performing the required operations, they can be self-operating or they can be attached with some other uncontrollable devices (e.g. doors, windows and gates) for controlling their functionalities. As these uncontrollable devices are used as an interface in the environment but they cannot be directly controlled through messages, for controlling them controllable devises are required to associate with them.

For the design of any SmE, it is required to identify the list of these devices with their positioning details. Also, the list of services, which are to be accomplished by using these devices, is created. Each service is associated to some devices in some relationship and against each service, SmE perform some certain functionality. At this step, it is also required to identify these relationships and the desired functionalities. Then, the overall constraints on the system are required to be identified so that they can be considered while modeling the SmE, such as the security constraints is to close all the entry points (e.g. main door, windows, rear door) when a smart home goes in “sleep mode” and the

safety constraint is to open all the entry and exit points in case of fire.

After this step, a clear picture of the SmE will be obtained. Caution is advised at this stage because reliable, secure and safe implementation of the system will closely follow these specifications.

The design specifications, the internal constraints and desired behavior of the BDSB systems are given bellow:

Design Specification

1. two doors (external and internal) are used for ensuring the security measures from the harmful access (direct access should not be possible) to the bank;

2. there is an isolated space between external and internal doors;

3. doors can be controlled from the outside and inside of the isolated space through the associated touch sensors installed at each side of the door (by sending the door-open request), so that the people can cross the door without being stuck;

Internal constraints and desired behavior

1. doors will remain open for a fixed time after opening and before closing so that the users can cross;

2. when one door is in the process of opening-and-closing and the same door-open request from the associated TS arrives, BDSB checks the state at which the request is received and accordingly performs the following action:

   (a) if the same door-open request arrives when the door is in the opening process, BDSB just holds this request and will not open the door again;

   (b) if the same door-open request arrives when the door is in the closing process, BDSB will re-open the door;

3. if one door is in the opening-and-closing process and the door-open request from the other door arrives, the BDSB will hold the request and wait for the closing of other door. As soon the other door will be closed, BDSB will open the requested door.

4.2.2 Step 2: Users Modeling

Users play a key role in the operations of SmE. According to their presence (observed from different sensing devices) and actions (performed on devices), SmE perform specific operations. For the identification and modeling of such requirements, a two steps process is adopted: goal modeling and behavior modeling. In goal modeling, the Goals, Actions and Roles of the users which they can achieve from SmE are described. Goals are the

set of objectives which can be performed/demanded by the users. For achieving these goals, users have to perform specific actions. Roles establish a relationship between the user actions and the environment, which allows the users for performing specific task according to the environment configurations. The flow of the task carried out in this step is shown in Figure 4.3.

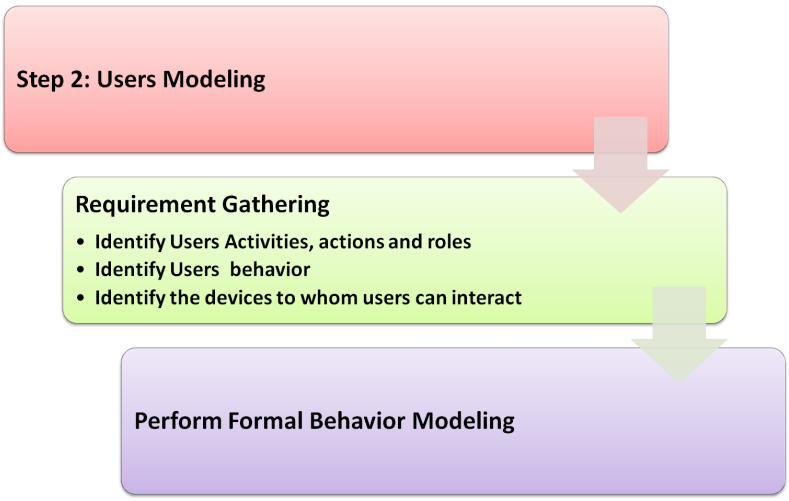

Figure 4.3. Users Modeling

The users have complex web of behaviors which they can adopt during the interaction with the system. In behavior modeling, the analysis of their all possible moves are identified and modeled by incorporating their organized goal information. Among different perspectives, some of the behavioral aspects, which we considered for users modeling in this thesis, are following:

1. How the users can interact with the system?
2. Which user actions are considerable for the system?
3. Where a user can be positioned after performing an action?
4. What are the set of possible user behaviors which they can adopt?

Some other aspects of the users, though not considered for this thesis, are the following: 1) Actions history of the users 2) Division of users on the basis of their roles

The following is the list of users' activities, behaviors and observations which are considered for the users' modeling in BDSB system:

1. user can access and return from the bank by crossing the doors;

2. users can press the associated touch sensors (at each side of the door) for opening the doors;
3. users can press touch sensors more than one time;
4. users observe the states of the doors and when a door is found open, they can act in following ways:
   (a) they may cross the door;
   (b) their mind may change and they stay there without crossing the door;
   (c) they cross the door, but sooner their mind may change and they cross-back and come to their previous location.
5. users can change their mind from the isolated space and exit from there without entering into the bank; similarly they can re-enter in the bank without exiting.

4.2.3 Step 3: Devices Modeling

Controlling and commanding the functionalities of electrical (low cost or smart) devices are main goals of SmE. These devices are of heterogeneous nature with some common and distinguish features (such as functionalities, commands, notification, states and others). The desired functionality from the relevant devices is accessed by inputting some specific commands or by interacting with them depending upon the type of the devices. For the sensor, the input is received by sensing the environment and its output is usually a notification message; whereas for other devices, the input can be a command and the output can be a physical operation. The input and output depend on the category of the devices; further the devices can be smart by having some internal constraints. These elements (input, constraints, output) are required to be gathered, organized and described at requirement gathering phase.

For the design and verification of complete interaction among SmE components, it is also required to model the attached devices at design time. The modeling of these devices can be performed by adopting interface (black box) and behavioral (white box) modeling schemes. Before modeling a device it is first required to collect their detailed relevant information, which includes the interface information – the commands (triggers) it may receive, the associated functionality (operation) it may perform, the constraints (rules) it has to follow, the states at which it will be at any time, the notifications which it sends after the completion of task – and behavioral information – the acceptance of specific commands on a particular state, the implementation of constraints, the operations which may be performed on that state after the satisfaction of constraints – of the particular devices. A graphical flow of the task carried out at this step is presented in Figure 4.4.

Touch sensors, door actuators, door sensors and obstacle detection sensors are used as controllable devices in BDSB. The modeling of each device is performed according to the

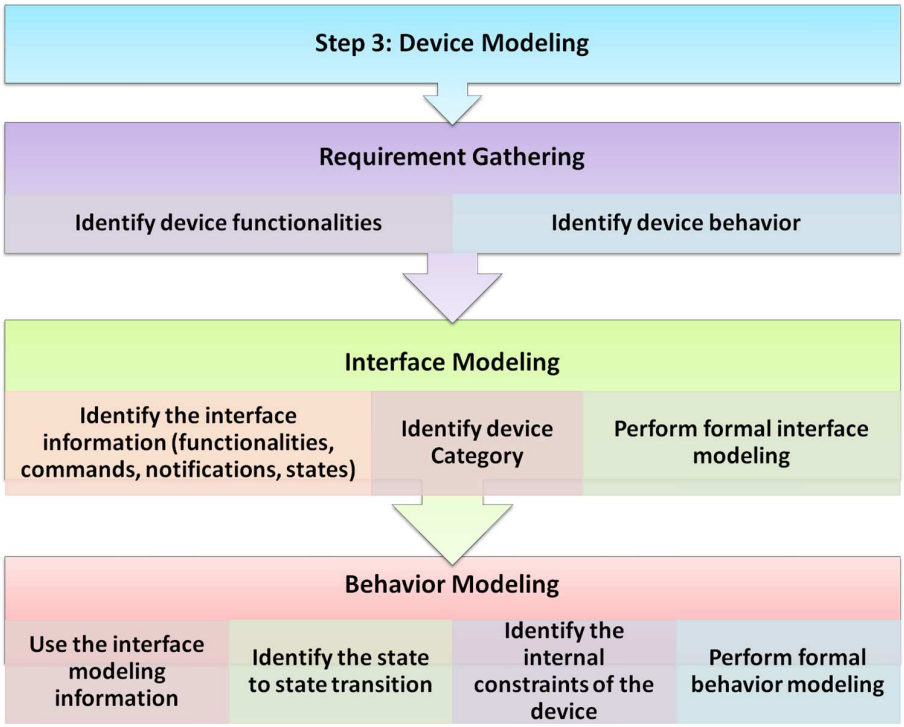

Figure 4.4. Devices Modeling

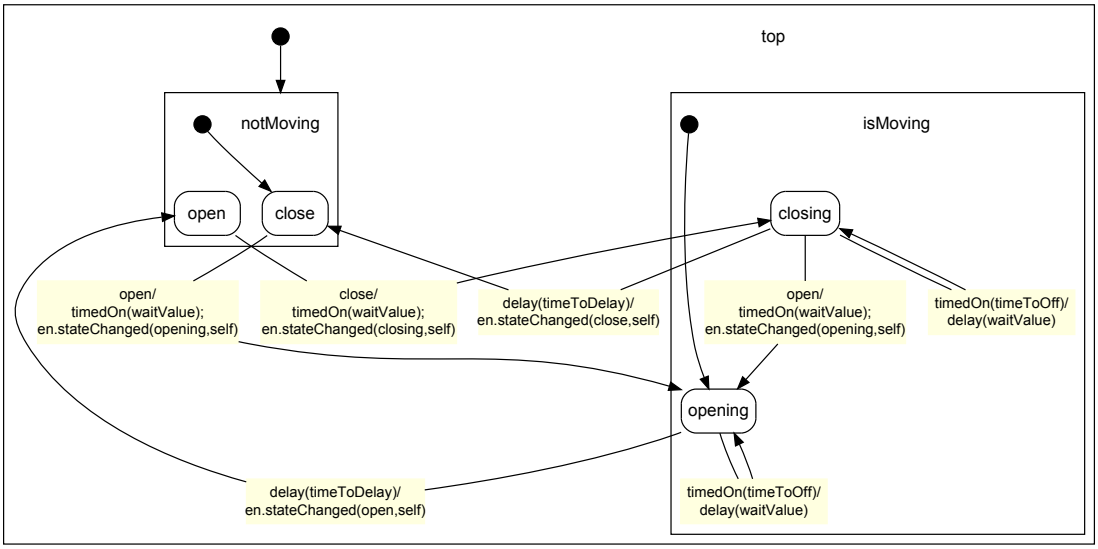

Figure 4.5. Statechart modeling of Door Actuator

activities specified in the methodology (from requirement gathering to their behavioral modeling); such as the door actuator component of BDSB system is described to have open-close functionality by which it provides force to open or close the door. The door actuator, at any specific time, can be in moving (opening-closing) or in non-moving (open-close) state. For activating the desired functionality, it accepts open or close command and, accordingly, performs its operation. It can also send the notification back after the state has changed. The behavioral modeling of door actuator, in Statechart format, is represented in Figure 4.5.

4.2.4 Step 4: Individual Device Verification

Every SmE (from simple to complex, based on the feature set offered) makes use of various devices of heterogeneous nature. The intelligence in SmE is provided by controlling the functionalities of these devices. Other components of SmE directly or indirectly interact with these devices. The modelling of these devices can be performed by adopting both black box and white box formalisms. The black box formalism will help other components to interact with them, whereas whenever their reliable behavior is required, the white box formalism is considered.

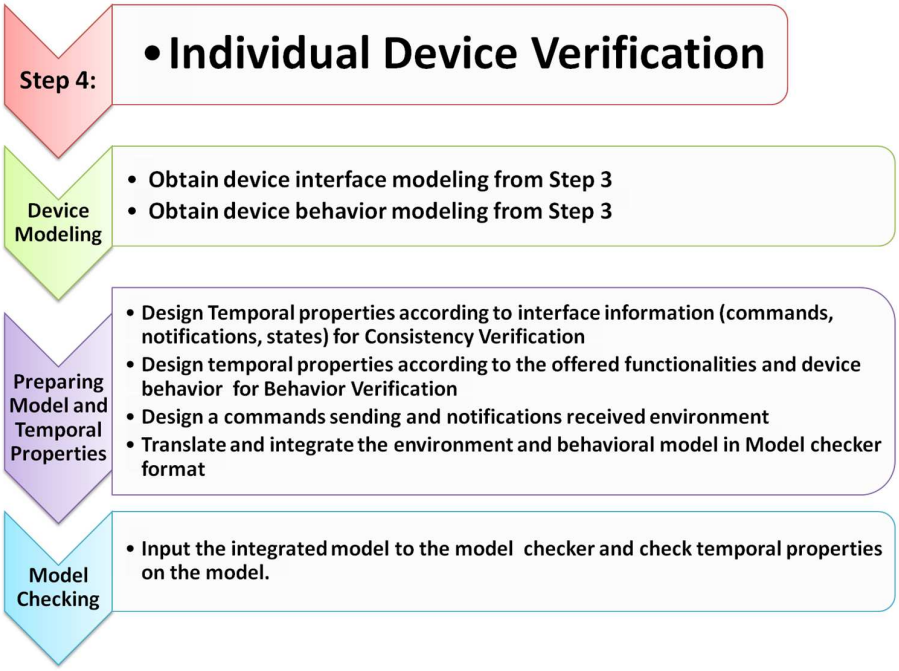

Figure 4.6. Individual Device verification

Before verifying the complete SmE, which may consist of a list of devices, it is best to verify the requirement accomplished behaviour along with the consistency verification

among the adopted black box and white box formalisms at early design stages. By this, at the time of entire system verification, the abstraction can be applied on the interaction, control strategy and the inner details of other components; and concentrating on the device behavior becomes challenging, which may cause the exploration of the graph (as a number of devices are used in SmE). In result, the drawback of using model checker technique, state-space-exploration problem, can be sufficiently controlled. The task carried out in this step is graphically represented in Figure 4.6.

The verification process can be performed with the use of model checker technique in which the model of each component is verified with the use of temporal properties related to the consistency and behavioral verification. For the individual component verification, it is required to convert the respective (device) model in the acceptable format of model checker by establishing a system through which the commands can be sent in any order, the acceptable commands can be accepted at that particular state and the associated functionality/operation be performed, the rest of the commands are ignored.

The requirements regarding the consistency (by considering one modelling formalism as a reference and comparing it with the other) and reliable behavior, according to the specifications, are firstly transformed into the format of temporal properties and then these properties are verified on the model with the use of model checker. In the case of unsatisfactory properties, the model is corrected and the verification process is repeated until all the properties are satisfied.

Some devices of BDSB system are Touch sensors, door actuators, door sensors and obstacle detection sensors. The implementation detail of this step is given in Chapter 5 Section 5.1

4.2.5 Step 5: Environment Modeling

In reality, users can observe the environment by seeing the current states of the concerning devices and accordingly interact with them for achieving the desired goals. But at design time, these features can be modeled by adding some extra computations through environment models. The environment models can update their configuration when any action or operation is performed by the concerning devices. Similarly, the environment model can be capable of registering the actions, locations and interactions of the users. At requirement gathering phase, the identification and listing of these computations, which are considered to be in the real environment, are required to be described. These descriptions help for the reliable modeling of the environment. The concerning features which are required to be considered for the environment modeling are graphically represented in Figure 4.7.

Considering the users' modeling at design time, it is suggested that the modeling of the environment component must also be done, as users may observe the environment configurations and accordingly interact with the system. For this, a mechanism can be designed which stores the state information of interesting devices so that the users' model

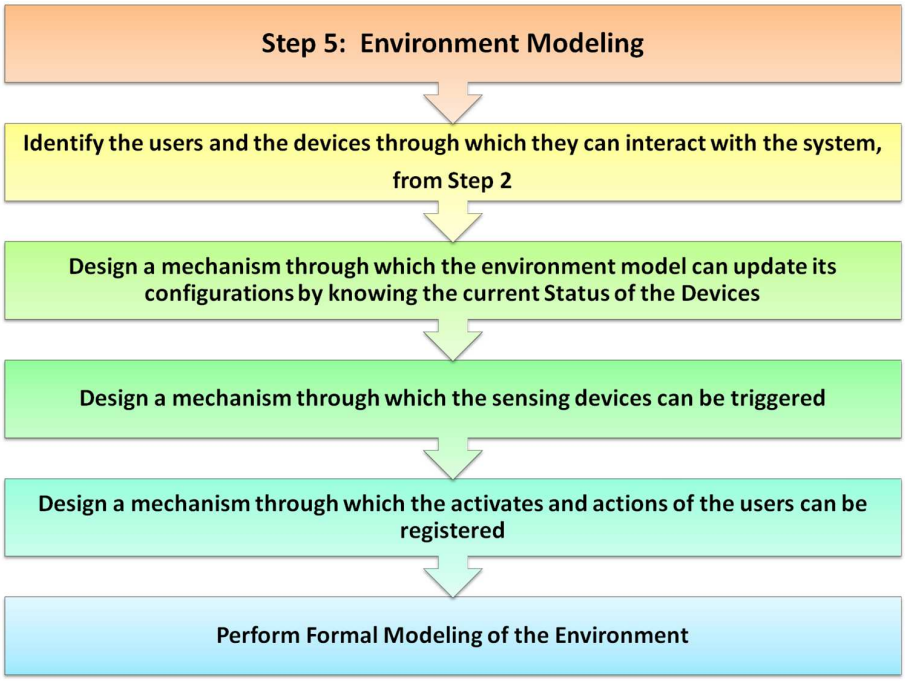

Figure 4.7. Environment Modeling

can observe the environment configurations at design time. As the devices model states change, the environment model updates the current state (of the particular device) with the new values. Similarly, the users' interaction with the sensors can be formalized with the use of environment modeling; the environment model can also register the activities of the user (so that the exact location of the users can be identified).

The users can view the states of the door, whether it is in open, close, opening or closing state; and accordingly perform some actions (e.g. cross the door, press the corresponding touch sensor). For designing such a real environment, an environment model is designed by having the ability to update it's configuration as soon as the doors change their states (taking advantage from State-Change-Notification message). Through this the users can have the latest configuration of the environment and can behave accordingly. Similarly, for knowing the proper location of users and accordingly providing access to the relevant devices, environment model registers the actions of the users. Additionally, the interaction with the obstacle detecting sensors can also be made through the environment modeling. All these features are modeled with the help of parallel Statechart formalisms.

4.2.6 Step 6: Control Algorithms Modeling

Control algorithms aid the computation in the SmE. For achieving a goal, the user performs an action which is forwarded to these controlling algorithms in the form of messages. According to these incoming messages, the current configuration of the whole system and the implemented rules, control algorithms make certain sophisticated decisions and send triggering messages to the associated devices for performing the required operations.

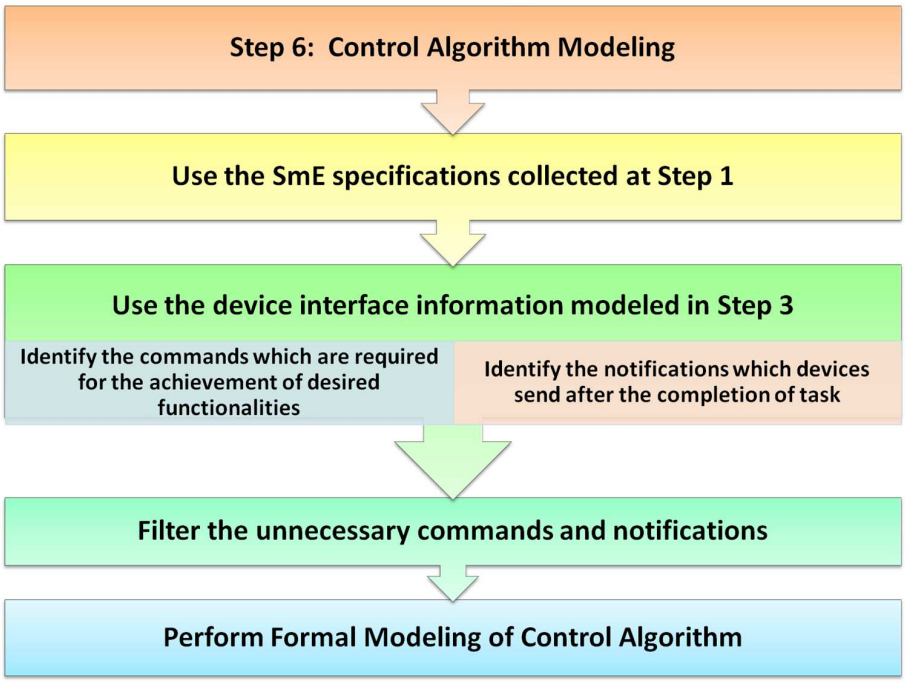

Figure 4.8. Control Algorithms Modeling

The desired behavior of SmE (listed in Step 1: SmE Specification Identification Stage), related to providing the required services, reliable behavior, security, safety and other constraints, is achieved through control algorithms. The control algorithms accomplish the required behavior by controlling the functionalities of the concerning devices. For an effective communication, control algorithms have to use the devices interface information (which are modeled in step 3: Devices Modeling). A graphical task flow of this step is given in Figure 4.8.

Door Lock Control (DLC) is an intelligent component of a BDSB system. It takes inputs from Touch Sensors (TS), Door Sensors (DS) and obstacle detection sensors (ODS), and according to the designed requirements, instructs the Door Actuators for opening/-closing the doors. All the computation requirements (mentioned in SmE Specification Identification Stage) are achieved through DLC. For achieving the desired computational

requirements (what to do when the door-open request arrives? when the requested door will be opened? when to send the acknowledgment?), different guards (constraints) are designed with the use of relational and logical operators. These guards work on the basis of incoming messages and the variable values.

4.2.7 Step 7: Temporal Properties Designing

It is important for any complex and critical system to ensure the successful modeling of all the desired behavior (related to the safety, security), functionalities and other constraints is performed. For the verification of these features, the modeling of the temporal properties is required so that they can be confirmed at the formal verification step. During the formal verification, some properties may likely be ignored due to system complexity. For reducing the chances of ignoring important properties, the requirements described so far are used. These requirements are further formulated by using the syntax and semantics of temporal logics.

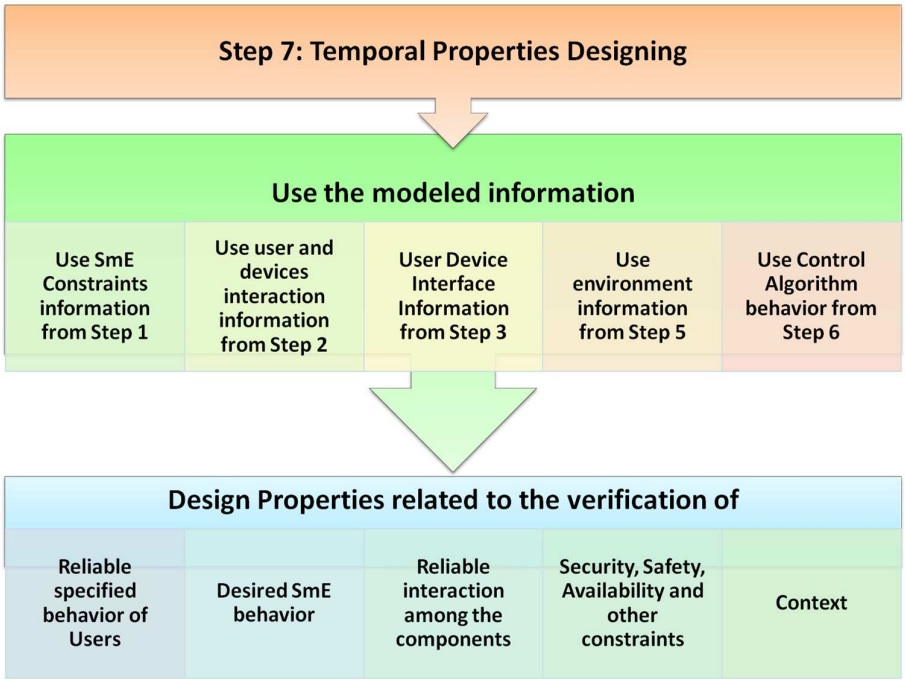

Figure 4.9. Temporal Properties Designing

The temporal logics are mostly used for the verification of the reachability of certain states, satisfaction of sequence, absence or existence of any predicate (at any state) and the boundary checking or the universality of any state or action. By using these features

of temporal logic, the properties can be designed by which the reliable specified behavior, safety, security and other constrains of the SmE can be verified. Tasks carried out in this step are graphically represented in Figure 4.9.

As mentioned, the requirements related to the reliable behavior of BDSB along with the safety, security and other constraints are formalized by using the syntax and semantics of UCTL temporal logic. The detailed description of these requirements with designed properties are given in Section 5.2.2: one of the requirements of BDSB system (in UCTL format) is that the external door will be opened when the user releases any touch sensor associated at each side of the external door.

$$EF_{\{extDoorCrossed\}} A \left[\top \ U_{\{\neg extDoorOpened\}} U_{\{T1ReleaseorT2Release\}} \ \top \right]$$

The touch sensor associated with the external door from the isolated space can only be accessed when user crosses the external door; therefore the first part of the property ensures that one user has crossed the door, now the door-open request of both sensor can arrive. The next path of the property is related to that scenario that the extDoorOpened request will not arrive until the associated touch sensors are pressed.

4.2.8 Step 8: Integrated SmE model

As control algorithms govern all the interaction among devices (and affect the environment), they receive a lot of messages (commands or notifications) from the connected devices. The devices models can send and receive nearly all possible messages related to their functionalities. But among these messages, some messages are of interest for the current system and should be modeled in control algorithms. The rest of the messages are useless for the current system, but it's a good practice that all the incoming messages must be received. If the modeling of all possible messages is performed in the control algorithms, then the size, complexity and ambiguities of control algorithms grow higher.

For curbing these issues, it is suggested to introduce a firewall around control algorithms which, at the initial level, checks the suitability of a received message and sends forward only those messages which concern the current system. Similarly, the received messages can have different parameters; therefore they can also be modified at this stage if required. This helps in optimizing the control algorithms: the processing load is reduced and the "lost-event" errors don't occur (during model checking) due to failure of acceptance at receiver's side.

Up to this stage, all the prerequisites for the modeling of SmE process are completed. Now it is required to convert them into the acceptable language of the model checker and then combine them so that a complete SmE model can be prepared. For the translation, the behavior models of the users, connected devices, environment, control algorithms, firewall (with messages filter and converter) are required, along with the proper abstraction and list of their instances (connected in the SmE). After converting them the whole

integrated SmE model is designed in the acceptable format of model checker. The task carried out in this step are graphically presented in Figure 4.10.

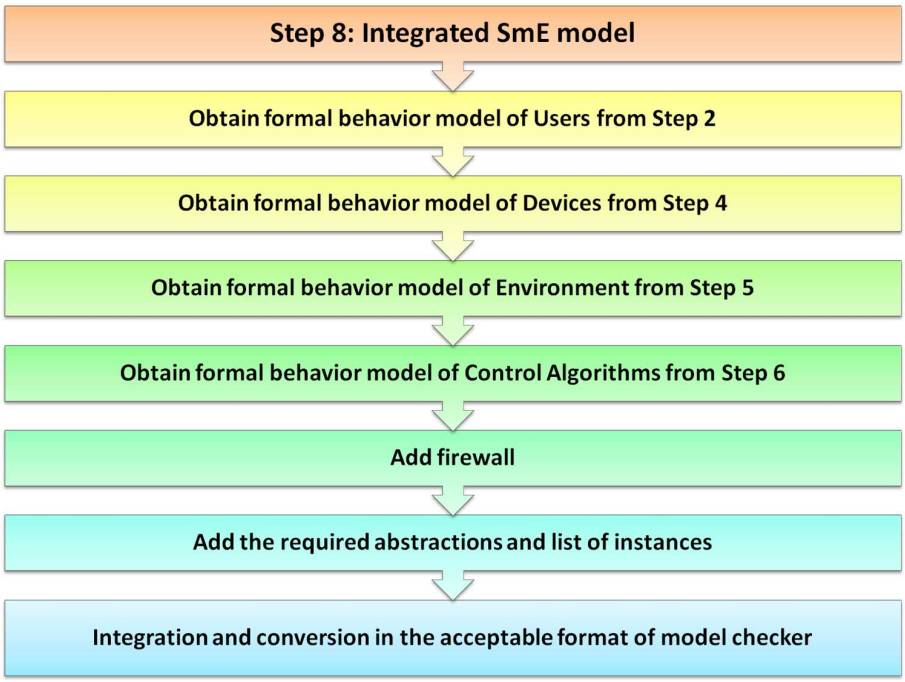

Figure 4.10. Integrated SmE model

The firewall is added so that the all the messages can be received and only the useful messages and notifications can be passed. Then, along with the firewall component, the individual behavior model of users, devices, environment and control algorithm are converted into the acceptable format of UMC. Further, the abstractions and instances information is added for completing the holistic integrated BDSB model.

4.2.9 Step 9: Formal verification of SmE Model

The whole integrated SmE model, in the acceptable format of model checker, designed at Step 8 is sent to the model checker, and the temporal properties (designed in Step 6) are verified on the model. On finding any unsatisfactory property, the SmE model is updated with the required modifications, and the verification process is repeated until all the properties are satisfied. The task carried out in this step are graphically presented in Figure 4.11.

All the temporal properties, included the one mentioned above, are verified on the BDSB model and the satisfactory results confirmed the successful exhaustive verification of our tested SmE, with the explicit focus on safety and security requirements.

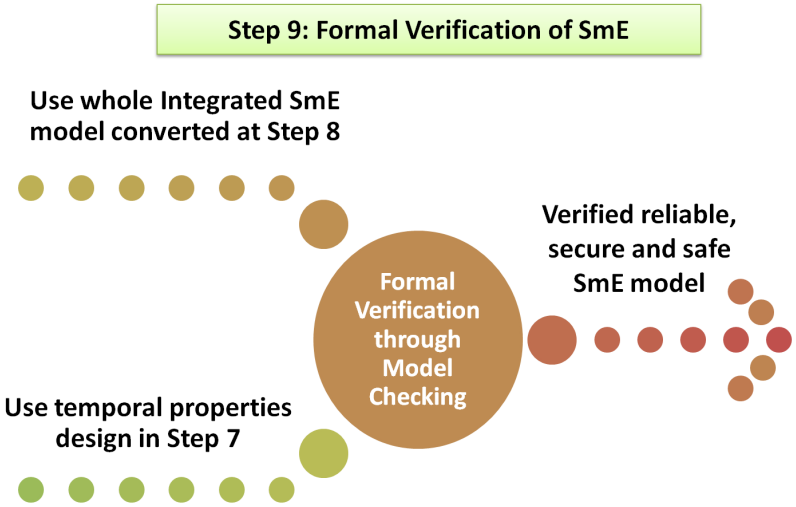

Figure 4.11. Formal verification of SmE Model

A brief description and implementation details of this step is presented in Chapter 5 Section 5.2.

4.2.10 Step 10: Development Phase

When all the properties are verified, it is implied that the SmE model is according to the specification and will behave reliably, surely and safely under the verified properties in all scenarios. It is time to safely start the development and implementation phases.

Chapter 5

Designed Techniques

A set of two techniques is designed and implemented by employing DogOnt, SCXML, UCTL and UMC as tools and follows the steps of proposed methodology (given in Chapter 4). In the first technique the reliable behavior of individual devices model, with respect to specifications, and the consistency among both black box and white box modeling formalizes are verified (step 4). Then, in the second technique, these verified behavioral models are integrated with other models and control strategy for verifying the entire SmE (step 9). The following sections briefly explain these techniques with examples (or case study) and their results.

5.1 Individual Device Verification

In the components verification phase, the consistency checking among their interface and behavioral modeling formalism is confirmed by automatically designing the temporal properties and, the behavioral verification with respect to the listed specifications are confirmed by manually designing the temporal properties.

The interface model of the device is known as “Device Interface Model” (DIM) and the behavioral model is known as “Device Behavioral Model” (DBM). In the device verification process, the consistency and behavioral verifications of the device are performed for resolving the following issues:

1. The DBM should perform the functionalities which are specified in DIM, so that, the device model can perform all its functionalities which it is capable of.
2. The DBM should perform the specified task against the specified commands which are mentioned in DIM, so that, all the associated task can be in the device model.
3. The DBM should send all the notifications back (as an acknowledgment about the completion of tasks) which are specified in DIM, so that the further activities, which are dependent on the arrival of notifications, can be performed.

4. The DBM should contain all states which are available in DIM, so that, the associated actions against such states can be performed by the device model.

5. The DBM should be modeled according to the specification, so that, the device model can perform its reliable functionalities.

6. The DBM should be modeled in such a way that a deadlock may never occur in any scenario.

5.1.1 Device Model Verification Technique

A technique is designed for the consistency and behavioral verification of devices and graphically represented in Figure 5.1. All the specifications are listed and written in an understandable natural language format. By considering these specifications, the interface modeling is performed with the use of DogOnt and the behavioral modeling is performed with the use of statecharts, know as Devices Statecharts (DSCs), in SCXML [145] format.

In the consistency verification phase, it is required to ensure that all the modeled commands, notifications and states (modeled in DIM) are incorporated in the DBM. As, the device can accept some specific commands at particular states (others are rejected on that state); for checking the existence of commands, by not knowing the command sequence structure in DBM. For achieving this constraint, a Closed Environment strategy is adopted, through which DBM can asynchronously accept commands and send notifications back after the completion of task. If the command/notification is implemented on a reachable state in the DBM, then the existence property of it gives a satisfactory result. Similarly, the reachability of the state can also be verified through Closed Environment strategy (if the state is in DBM but can not be reached, then its existence is meaningless, therefor reachability verification is performed).

The Environment Designer component, represented in Figure 5.1, automatically generates “Environment Generate Commands” (EGC) and “Environment Receive Notifications” (ERN), by considering DogOnt, in a form acceptable by Model checker (which is UMC in our case), graphically represented in Figure 5.2. EGC is responsible for generating all the commands specified in DogOnt against the device being currently checked, whereas ERN is responsible for accepting all the notifications specified in DogOnt against it.

The EGC and ERN are sent to Model Builder (MB), where MB performs the following activities:

1. It automatically obtains the DBM, which is in the form of SCXML, from “Library File” container.

2. It automatically converts the SCXML model into the format acceptable by model checker (in our case UMC)

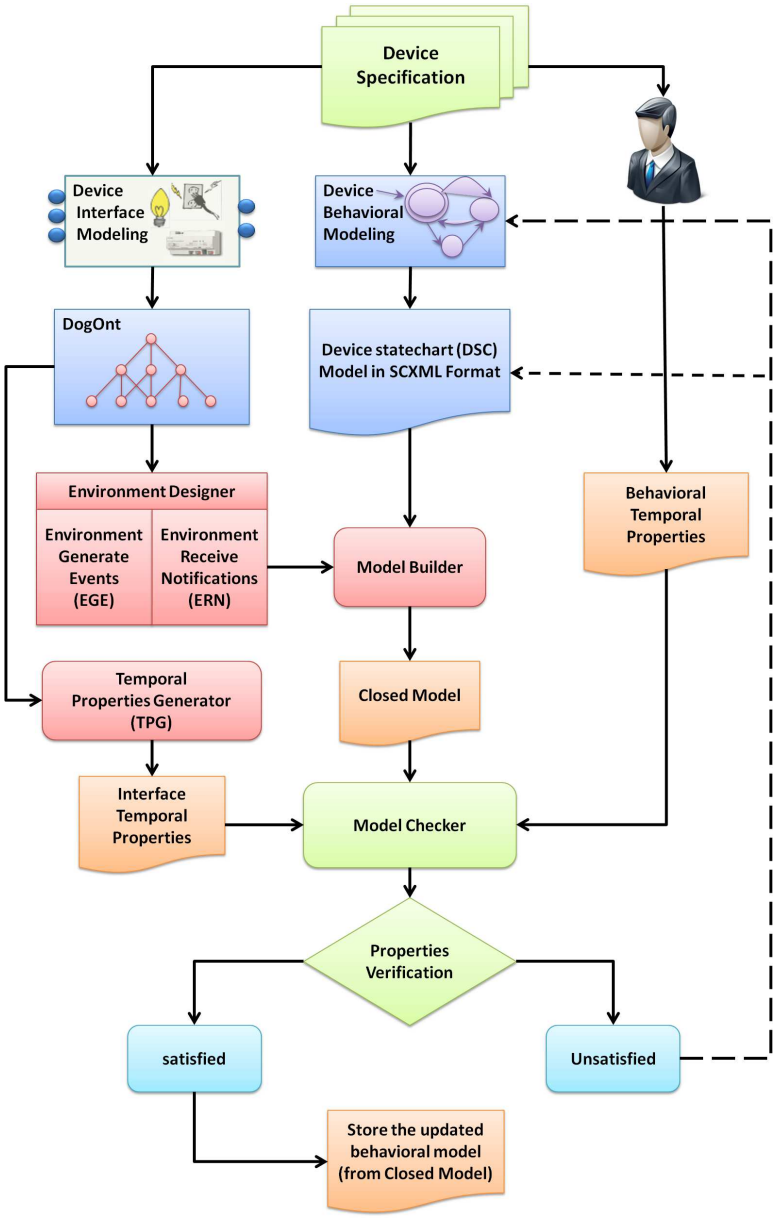

Figure 5.1. Device Model Verification Technique

3. It automatically combines the statecharts of EGC, ERN and the converted SCXML model.
4. It automatically adds some abstractions rules.
5. It, in last, saves all the gathered and designed information in a file, known as Closed Model.

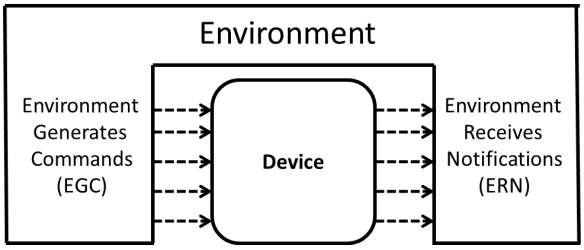

Figure 5.2. Closed Environment Strategy

Temporal Property Generator (TPG) is responsible for automatically generating the Temporal properties, in the form of UCTL, to verify the consistency between the DIM and DBM models. As, there is no direct way to match them, because both are used for different purposes, but it is important that both have same information. For this, TPG automatically designs three types of action-and-state based temporal properties by considering the information available in DogOnt. The properties are related for checking that the system can send all the mentioned commands and the device can received these associated commands, similarly after completing the task, the device send the notifications (as an acknowledgment) and the system can accept them, along with the verification of the reachability of the specific states.

The temporal properties, related to “sending” of messages, are required because of the possibility of some notifications being available in DogOnt but not being modeled in DSCs. The “accepting” message properties are required because of the possibility of DSCs missing to model some functionality which is accessible through a specific command. The last type of temporal properties is responsible to check the reachability of states, which are modeled in DogOnt.

The closed model and the temporal properties, designed by the TPG, are verified through the model checker. On finding any unsatisfactory property, it is required to update the DSCs by fixing the problem. When all the properties are satisfied, the verifier may manually generate temporal properties for verifying the correct behavior of the device. In the behavior verification phase, these behavioral properties are designed manually, by considering the device specifications. In the case of any incorrect property, it is required to update the DSC by fixing the issue and regenerating the close model. The behavioral properties are verified until all the properties (behavioral) are satisfied. After this process, the DSC is verified and can be used in the driver development process or for verifying the system through simulation, emulation, model checking and others.

Dimmer Lamp: A Case Study

A case study of a dimmer lamp is presented here in which consistency and behavioral issues are found. We consider the interface modeling (through DogOnt) of the Dimmer

Lamp, as presented in table 5.1, in which all the, direct and inherited, functionalities and states which a dimmer lamp can hold are summarized.

Dimmer Lamp	has Functionality	OnOff Functionality	has Command	off Command	real Command Name	off	Control Functionality
				on Command	real Command Name	on	
		Light Regulation Functionality	has Command	StepUp Command	real Command Name	stepUp	
				StepDown Command	real Command Name	step Down	
				set Command	real Command Name	set	
					command Param Name	"value^^ Object"	
		Level Control Functionality			max Value	"100"	
					min Value	"0"	
		State Change Notification Functionality	has Notification	State Change Notification	notification Name	state Changed	Notification Functionality
					notification Param Name	"new State^^ State"^^ Literal	
		Query Functionality	has Command	Get Command	real Command Name	getState	Query Functionality
					return Type	Object	
	hasState	*OnOff State*	has Stae Value	OnState Value	real State Value	on	Discrate State
				OffState Value	real State Value	off	
		Light Intensity State	has Stae Value	Light Intensity State Value	real State Value	" 1 Literal"	Continuous State
	-	Device Interface Detail			+		

Table 5.1. Dimmer Lamp details available in DogOnt

With the use of information, provided in table 5.1, “Environment Generate Commands” (EGC) and “Environment Receive Notifications” (ERN) automatically generate the sending commands and receiving notifications models (in the acceptable format of model checker) from DogOnt. Then “Model Builder” (MB) obtain DSC of the device and automatically convert the DSC in the format acceptable by the model checker, it then combines this converted model with the models generated by EGC and ERN, further it automatically adds the required abstractions and generates the instances of these model. After this, the combined model is ready for verification, we save this model in the file, know “Closed Model”, as represented in Figure 5.3 and passed it to the model checker.

“Temporal Property Generator” (TPG) automatically generates all the related interface properties from DogOnt. The designed properties related to functionalities and notifications are of two types, one is used for checking the “sending” of the commands/notifications, whereas the other is related to the “accepting” of these commands/notifications.

```
Class State is
end State;
Class EGC is
  Vars: RandomValue:int=35
  State top = E
  Transitions:
    E -> E {-/DimmerLampInstance.set(RandomValue)}
    E -> E {-/DimmerLampInstance.stepDown()}
    E -> E {-/DimmerLampInstance.stepUp()}
    E -> E {-/DimmerLampInstance.off()}
    E -> E {-/DimmerLampInstance.on()}
end EGC;
Class ERN is
  Operations: stateChanged(newState:State)
  State top = N
  Transitions:
    N -> N {stateChanged(newState)/}
end ERN;
Class DimmerLamp is
  Operations: on(), off(), set(value:int), stepUp(), stepDown()
  Vars: lightIntensity:int=50, lightStep:int=10
  State top = off, on
  State on = lightIntensityState
  Transitions:
    off-> on{on()/ }
    off-> lightIntensityState{set(value)/ lightIntensity:=value}

    on -> lightIntensityState{-/}
    on-> off{off()/ }

    lightIntensityState ->  lightIntensityState{stepUp() /
      if (lightIntensity < 100 )then
      {lightIntensity :=  lightIntensity + lightStep} else {lightIntensity :=  100}; }

    lightIntensityState ->  lightIntensityState{stepDown() /
        if (lightIntensity > 0)then
        {lightIntensity :=  lightIntensity - lightStep} else {lightIntensity :=  0}; }

    lightIntensityState ->  lightIntensityState{set(value)/ lightIntensity:=value}
end DimmerLamp
Objects:
ec: EGC
en: ERN
DimmerLampInstance: DimmerLamp
Abstractions{
 Action $1($*) -> $1($*)
 Action $1 -> sending($1)
 Action accept($1) -> accepting($1)
 Action lostevent($1) -> discarding($1)

 State inState(DimmerLampInstance.lightIntensityState) -> LightIntensityState
 State inState(DimmerLampInstance.off) -> offState
 State inState(DimmerLampInstance.on) -> onState
 State DimmerLampInstance.lightIntensity < 0 -> underFlow
 State DimmerLampInstance.lightIntensity > 100 -> overFlow
 State DimmerLampInstance.lightIntensity >= 0 and DimmerLampInstance.lightIntensity
          <= 100 -> inRange
}
```

Figure 5.3. Closed Model of Dimmer Lamp

```
--Action Properties
  --the acceptance of all the commands in DSC
    EF {sending(stepDown)} true
    EF {sending(stepUp)} true
    EF {sending(set)} true
    EF {sending(off)} true
    EF {sending(on)} true
  --
    EF {accepting (stepDown)} true
    EF {accepting (stepUp)} true
    EF {accepting (set)} true
    EF {accepting (off)} true
    EF {accepting (on)} true

  --the generation of all the notifications in DSC
    EF {sending(stateChanged)} true
    EF {accepting(stateChanged)} true

--State Properties
  --the reachability of all the states in DSC
    EF (offState)
    EF (onState)
    EF (LightIntensityState)
```

Figure 5.4. Temporal Properties for Interface Verification

```
E [true {not accepting(off)} U {accepting(on) or accepting(set)} true]
E [true {not (accepting(stepDown)or accepting(stepUp))} U {accepting(on) or
  accepting(set)} true]
EF (not underFlow)
EF (not overFlow)
EF (inRange)
```

Figure 5.5. Temporal Properties for Behavioral Verification

The properties related to the reachability of states are also generated by TPG. The automatically generated properties related to the interface verification of Dimmer Lamp are presented in Figure 5.4.

Results of Dimmer Lamp

The automatically generated properties, presented in Figure 5.4, are verified on the Closed model. It is found that the sending property related to the “stateChanged” notification is not satisfied, which shows that in designing DSCs this notification is ignored and not modeled in DSCs of Dimmer Lamp. The error is fixed by modifying the DSC and the verification process is repeated, which shows the verification of all interface properties.

The behavioral properties related to the verification of Dimmer Lamp are manually formalized, as shown in Figure 5.5, according to device specifications. When these properties are verified on the model, it is found that the last three properties related to bound

checking (as light intensity of Dimmer Lamp must be in-between 0% to 100%) are violated. The model is analyzed, and the condition *lightIntensity* $<$ *100* is replaced by *lightIntensity* + *lightStep* <= *100*; similar action is taken for the decrement condition, too. After this the behavioral properties are verified again and all the properties are found satisfactory.

The modifications made in the DSC and accordingly the Closed model is regenerated and verified. Now, the both (interface and behavioral) modeling formalisms will be consistent and according to the specification (mentioned in activity 1). After this process, the following goals will be achieved regarding the specific verified device:

1. All the interaction commands which are modeled in DogOnt, for triggering the specific task, are also modeled in DSC.

2. All the notifications which are modeled in DogOnt, as acknowledgments about the task completion, are also modeled in DSC.

3. All the states which are modeled in DogOnt, for performing specific actions, are also modeled in DSC.

4. All the functionalities which are modeled in DogOnt, for knowing the capabilities of the device, are modeled in DSC.

5. The DSC model is also verified for the existence of any deadlock.

5.1.2 Experiments and Results

The feasibility of the "Individual Device Verification" process is checked by selecting and verifying thirteen devices among the 143 devices modeled in dogOnt. During this verification process the consistency errors and the behavioral issues are found in some DSCs. In table 5.2 the interface information of these devices are presented, with the additional information of the results of the Automatically designed and satisfied interface properties (are shown in last two columns of the table 5.2), by the model checker.

5.2 SmE Verification

The desired SmE model is verified by integrating the behavioral models of each components, control strategy, the required firewall with the desired abstractions. The technique, based upon the steps of proposed methodology, is presented in the following subsection with the results of the BDSB case study (presented in Chapter 4).

Device	Number of External Commands	Number of DogOnt Commands	Number of DogOnt Notifications	Number of DogOnt States	Number of Explored States (max)	Number of Automatically designed TPs	Number of Satisfied Properties
Button	1	0	3	2	16	8	8
Dimmer Lamp	0	5	1	3	417	15	13
Door Actuator	4	2	1	4	65	10	8
Door Sensor	2	0	3	2	16	8	8
Infrared Sensor	2	0	3	2	16	8	8
Mains Power Outlet	0	6	1	2	30	16	12
On Off Switch	1	0	3	2	12	8	8
Shutter Actuator	2	3	1	5	50	13	11
Simple Lamp	0	2	1	2	14	8	8
Smoke Sensor	2	0	3	2	16	8	8
Toggle Relay	0	1	3	2	12	10	10
Touch Sensor	2	0	3	2	16	8	8
Window Actuator	4	2	1	4	65	10	8

Table 5.2. List of Verified Device Models (DSCs)

5.2.1 Designed Technique

A graphical view of the technique is represented in Figure 5.6. It works by adopting the following activities:

1. SmE and its related components requirements are organized according to the operational flow and various Steps of the proposed methodology;
2. the behavioral components of SmE are collected;
3. according to the specifications, the control strategy is designed in the form of SCXML statechats;
4. the firewall component (for filtering and converting the messages) is represented in SCXML semantics;
5. the behavioral models of the SmE components and firewall are converted in the acceptable format of model checker;
6. the required abstractions with the device instances information are queried from DogOnt and added at the end of converted model (in the acceptable form of model checker);
7. the computation requirements in the form of properties are formalized by adopting the following steps:
   (a) according to the modeled requirements, the possible computational properties are identified;
   (b) for designing the properties, the system configurations (such as the information of all the associated instances of devices with their location, states, functionalities, commands, notifications and others) are queried with the use of DogOnt;
   (c) the Statecharts modeling of the corresponding components are used for querying the sequences of commands, notifications and states;
   (d) properties are designed based on above mentioned information, by using the syntax and semantics of temporal logic acceptable by model checker (UCTL in our case);
8. the designed properties and the complete SmE model are passed to the model checker (UMC in our case), which verifies these properties on the model and reports about their satisfaction:
   (a) in the case of finding unsatisfactory properties, the corresponding behavioral models are updated with the required modifications;

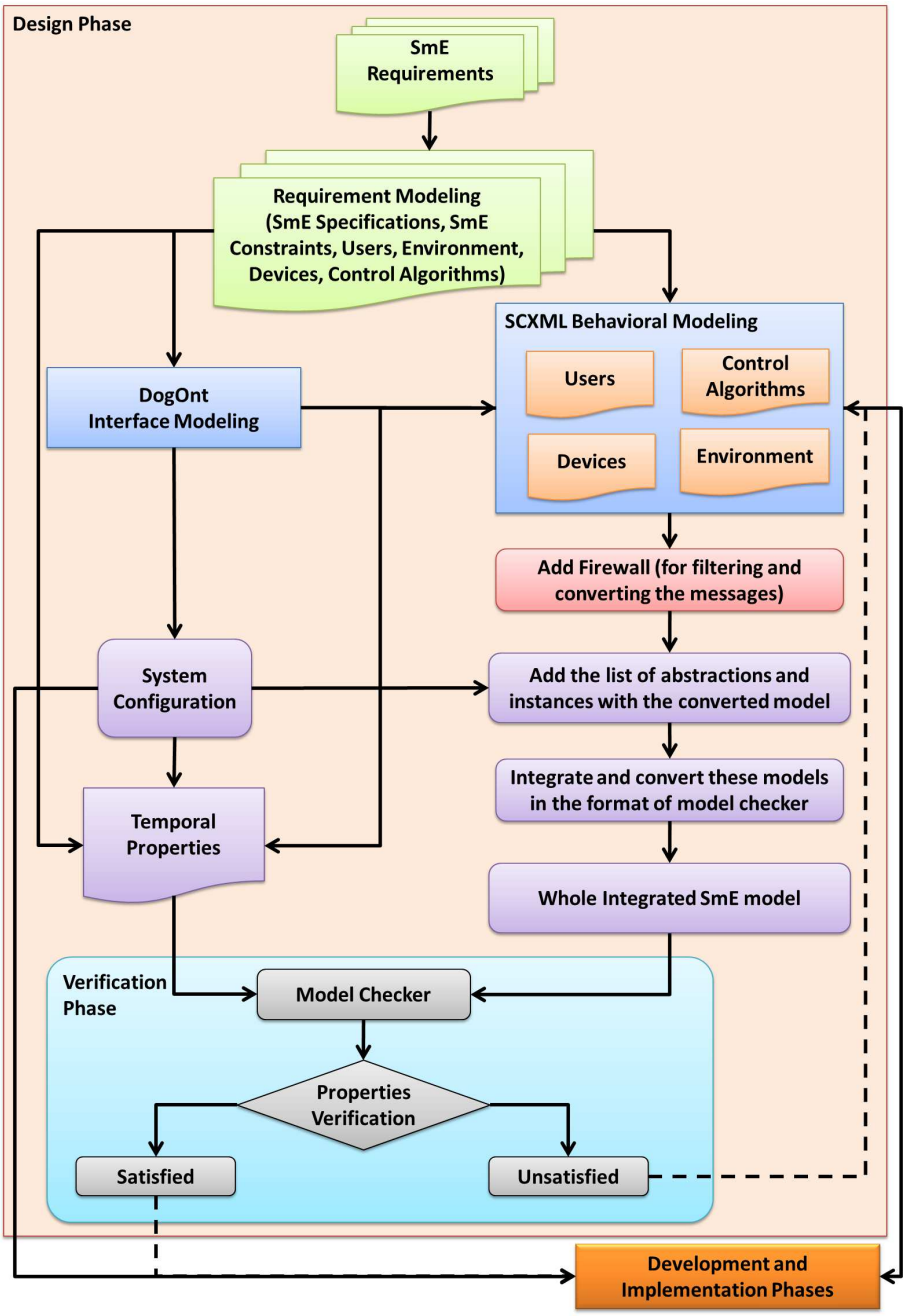

Figure 5.6. Designed Technique

(b) the verification process is repeated until all the properties are satisfied

When all the properties are satisfied, then the system can be declared as reliable, safe and secure, and will behave well according to the verified requirements in all scenarios.

As a result, the implementation phase can be started.

5.2.2 Experiment and Results

In this Section, the requirements related to the safety, security and reliable behavior of BDSB and its related components are formalized according to the categories (users' behavior modeling, users and their interaction modeling with the devices, device modeling, devices interaction and their control modeling and context modeling), by considering the message exchange behavior of BDSB and its components. These properties are then specified in UCTL format. All the properties are individually verified on BDSB model. The abstracted evolution graph (generated by UMC) of BDSB model consists of 2,79,119 states with the depth at 30 levels. The time taken for verifying each property was usually less than a minute in the on-line version of UMC. In Table 5.3, the reference of these properties is given with their evaluation time, the number of states and computations fragments generated for evaluating them. During the verification process at first stage, it was found that the designed model did not satisfy all the properties. UMC provides an error trace tree through which the errors have been located and the model was updated by fixing the bugs. The verification process has been repeated until all the properties were proven TRUE against the BDSB model.

Properties related to Users behavior

The user modeling is performed according to the specifications; all the users can enter the bank by crossing the external door, the isolated space and the internal door. It is also possible that users may change their mind and stay out without crossing the external door. Therefore, path 'Existence' quantifier is used in the property instead of 'All' quantifier for the verification. Similarly, users' mind may change and they may go back from the isolated space without crossing the internal door. For verifying that users can access the places, the following set of state properties (by using state abstraction) is formalized.

P1) $EF_{(u1AtOutsideTheBank)}$

P2) $EF_{(u1AtIssolatedSpace)}$

P3) $EF_{(u1AtInnerSideOfTheBank)}$

P4) $EF_{\{extDoorCrossed\}} \top$

P5) $EF_{\{extDoorCrossed\}} \top$

Property	**Evaluation Time** ***(in Sec.)***	**States Generated**	**Computations Fragments Generated**
P1	< 1 ms	2	2
P2	< 1 ms	63	110
P3	0.33	3778	7461
P4	< 1 ms	62	52
P5	0.48	3791	3826
P6	< 1 ms	2	2
P7	0.03	389	593
P8	0.03	388	384
P9	0.50	3940	4310
P10	< 1 ms	2	2
P11	0.02	286	310
P12	0.51	4310	5511
P13	6.91	77397	81725
P14	0.50	3936	3993
P15	0.74	6252	6782
P16	< 1 ms	37	63
P17	0.26	2770	3039
P18	0.26	2770	2976
P19	6.76	77083	80043
P20	0.08	821	1005
P21	0.83	6935	7966
P22	0.07	819	819
P23	0.14	1388	2169
P24	0.52	4117	4281
P25	0.74	6252	6782

Table 5.3. The properties with their evaluation details

Properties related to actions performed by the Users

For achieving any goal, users have to perform some action. To know that users can press and release the respective touch sensors, the following set of properties is formalized. Although all the users can access the outside touch sensor of external door, the other sensors (T2 and T3) can only be accessed when the user has crossed the external door, whereas T4 can only be accessed when user has also crossed the inner door. Therefore, 'Existence' quantifier is used with the properties of other touch sensors.

P6) $AF_{\{T1Release\}}\top$

P7) $EF_{\{T2Release\}}\top$

P8) $EF_{\{T3Release\}}\top$

P9) $EF_{\{T4Release\}}\top$

Properties related to Users and Device Interaction

The external door will be opened when the user releases any touch sensor associated at each side of the door. Same will happen with the inner door. The following set of properties is used to verify such type of users' interaction with the devices.

P10) $A\left[\top\ _{\{\neg extDoorOpened\}}U_{\{T1Release\}}\ \top\right]$

P11) $EF_{\{extDoorCrossed\}}$
$E\left[\top\ _{\{\neg extDoorOpened\}}U_{\{T2Release\}}\ \top\right]$

P12) $EF_{\{extDoorCrossed\}}$
$A[\top\ _{\{\neg extDoorOpened\}}$
$U_{\{T1ReleaseorT2Release\}}\ \top]$

P13) $EF_{\{innerDoorCrossed\}}$
$E\left[\top\ _{\{\neg innerDoorOpened\}}U_{\{T3Release\}}\ \top\right]$

P14) $EF_{\{innerDoorCrossed\}}$
$E\left[\top\ _{\{\neg innerDoorOpened\}}U_{\{T4Release\}}\ \top\right]$

P15) $EF_{\{innerDoorCrossed\}}$
$A[\top\ _{\{\neg innerDoorOpened\}}$
$U_{\{T3ReleaseorT4Release\}}\ \top]$

Properties related to Safety Constraints

One of the safety constraints is to ensure no user is stuck inside the isolated space. In any case, the user may exit the space by either entering inside the bank of exiting out. The following set of properties is used to verify this type of safety constraints.

P16) $AF_{\{T1Release\}} AF_{\{DoorResponse(open,DAExt)\}} \top$

P17) $EF_{\{T2Release\}} AF_{\{DoorResponse(open,DAExt)\}} \top$

P18) $EF_{\{T3Release\}} AF_{\{DoorResponse(open,DAInner)\}} \top$

P19) $EF_{\{T4Release\}} AF_{\{DoorResponse(open,DAInner)\}} \top$

Properties related to individual Devices

When the command for opening the door is passed to any door actuator, it will open the respective door as a result. These properties are used to verify the functionalities of the door actuators that, when they receive the open command, after opening the door, they will also close it.

P20) $AF_{\{OpenExtDoor\}}$
$AF_{\{DoorResponse(close,DAExt)\}} \top$

P21) $EF_{\{OpenInnerDoor\}}$
$AF_{\{DoorResponse(close,DAInner)\}} \top$

Properties related to Security Constraints

Ideally, both of the doors should not be opened at a same time, the open door must be closed first and then the other requested door will be opened.

P22) $A[\top\ _{\{\neg DoorResponse(open,DAInner)\}}$
$U_{\{DoorResponse(close,DAExt)\}}\ \top]$

P23) $EF_{\{extDoorCrossed\}}$
$A[\top\ _{\{\neg DoorResponse(open,DAInner)\}}$
$U_{\{DoorResponse(close,DAExt)\}}\ \top]$

P24) $EF_{\{innerDoorCrossed\}}$
$A[\top\ _{\{\neg DoorResponse(open,DAExt)\}}$
$U_{\{DoorResponse(close,DAInner)\}}\ \top]$

Properties related to Context Awareness

The users can access the touch sensors only when they are at a proper location. When users are inside the bank, they can come out from the bank by pressing the touch sensor attached at the inner side of the bank.

P25) $EF_{\{innerDoorCrossed\}}$
$A[\top\ _{\{\neg innerDoorCrossed\}}U_{\{T4Release\}}\ \top]$

5.3 Discussion

The Table 5.3 shows the temporal values of verification of various tested properties. The average time for verifying all the 25 properties is 0.79 sec., with the standard deviation 1.83. As a general rule, the superficial properties (for which the on-the-fly model checker didn't have to go deeper inside the system for verification and a smaller number of states are generated) are verified in relatively lesser time, such as P1, P2, P4, P6, P10 and P16 (takes less than 1 millisecond ($<$ 1 ms)). Whereas the complex properties (for which the on-the-fly model checker had to go deeper inside the system for verification and a larger number of states are generated) are verified using more time, such as P13 and P19.

The proposed design time verification methodology, aided by user behavior modeling, device modeling, environment/context modeling, control algorithm modeling, and their interaction modeling, has demonstrated successful results for verifying the correctness, reliability, safety, security and desired behavior of SmE systems. The methodology proceeds sequentially from requirement listing to modeling and formal verification. The probability of missing any properties has been efficiently controlled by requirement listing. The methodology is implemented through the designed technique and implemented on a small but not so simple real life SmE system. The first run of verification process did not achieve all the properties as satisfactory against the model. After appropriate modifications to the model, it was then proven to conform to design requirements. This verified model can be used safely at the implementation phase.

Chapter 6

Achievement of High Level Goals

Modern Smart Environments (SmE) are equipped with a multitude of devices (e.g. sensors, actuators, lamps or TV) aimed at intelligent services. The services are associated with a multitude of resources and can relate to acquiring the functionalities of a single device or a group of devices. With the growth in the heterogeneous nature of devices regarding their controlled (various among manufactures) and offered functionalities, the issue of effectively managing the SmE is being raised. The trend of device-centric management [33–35] is going to shift with abstract modeling approaches [35, 37–39]. These approaches are aimed at providing the High-Level description of the goals of SmE services for interacting and controlling the functionalities of the associated devices.

It is imperative to note that the accomplishment of device functionality is subject to their required states [146, 147]. Thus a goal of *illuminating the bedroom* can be achieved when a lamp placed inside the bedroom is in *ON* state; whereas, *sleeping mode* can be achieved by rendering all the lamps and cooking appliances in *OFF* state, entrance doors in *LOCK* state, windows in *CLOSE* state, bedroom window shutter in *Half-Open* state, burglar alarms in *ACTIVE* state and the room temperature adjusted to $25\,^{\circ}\mathrm{C}$.

The devices are wide-ranging: simple (like lamp) to complex (like TV) based on their inherent functionalities. A complex device can have a composite state at any point in time according to the functionalities offered. For example, a switched *ON* TV, with volume level at *60%* and channel value at *7*, represents a composite state. The composite states can be modeled as parallel (or concurrent) sub-states of a device. In the remainder of chapter, the term *state* refers to the composite state of a single device.

Before serving a goal, a device can be in any state which is referred as *source state*. Later, when it successfully fulfills a goal, it reaches in *destination state*. Thus, a goal consists of a list of the corresponding devices along with their desired destination states. The deterministic process of traversing the states (from source to destination) of a single device is called an *evolution* and of all the devices involved in fulfilling the goal as a *global evolution*.

The evolution consists of *transition(s)*, as depicted in Figure 6.1. Depending upon

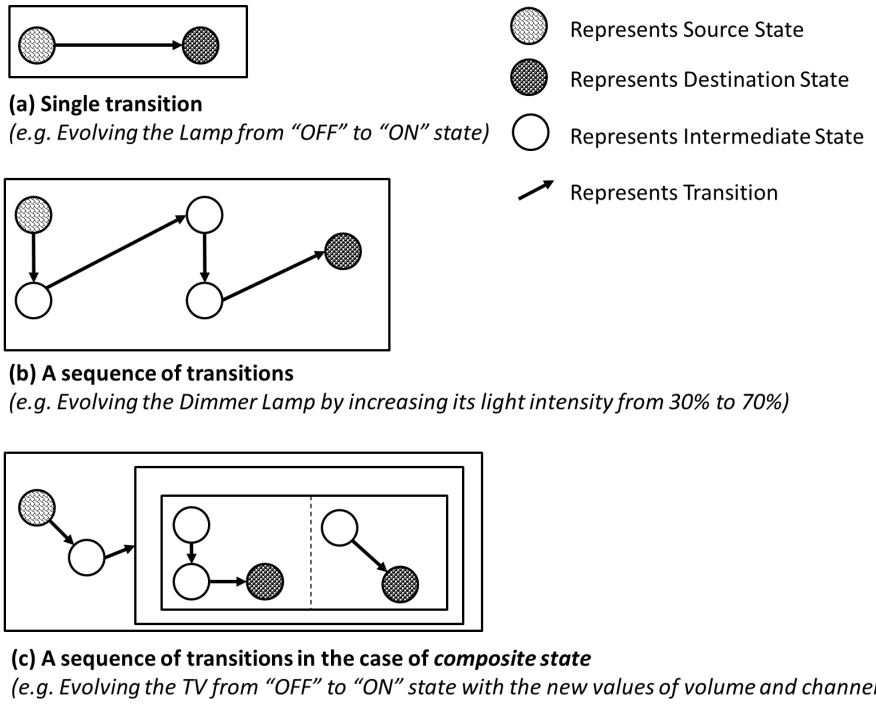

Figure 6.1. Evolution from Source State to Destination State

the source state, the destination state and the nature of the devices, the transitions can be entirely sequential (parts a & b of Figure 6.1) or partially-ordered (parallel state in part c of Figure 6.1). A transition consists of a *command* (to trigger the devices), against which the device performs its specified functionality and sends back (one or more) *notification(s)* (acknowledgment about the status of the assigned task). On receiving specified notification(s), the following command may render the device in the next state.

For the automatic generation of global evolution – in accordance with the goals – the task of finding certain transitions and their correct sequence for each device is detailed and complicated due to possible variance in the specific commands and their sequences according to the source and destination states of each device. This chapter addresses the issue of automatically finding and enforcing the correct sequence of commands required for the global evolution to achieve the high-level goals. For this, a comprehensive methodology is proposed which can be adopted across different SmE. The methodology works at two levels: design-time and runtime, and is applied on top of the *Domotic Effects* framework [38]. The implementation and experimentation details encourage the applicability and effectiveness of the proposed approach for the SmE.

The remaining chapter is organized into following sections: Section 6.1 provides an overview of the existing literature; Section 6.2 formally defines the addressed problem;

Section 6.3 presents an example carried out in this chapter; Section 6.4 explains the proposed methodology with the implementation details; Section 6.5 describes the experimentation; and Section 6.6 contains the discussion on the results.

6.1 Related Work

The related work is divided into two main areas: Goals Modeling and Evolution Finding. The related work regarding the goals modeling is presented in Section 6.1.1 and the discovery of evolutions is presented in Section 6.1.2.

6.1.1 Goals Modeling

The Smart environments (SmE) offer the service delivery in intelligent manners. These services are accomplished by transforming the states of their associated devices into their desired destination states. The modeling of these services at abstract level can be referred as High-Level goals modeling. The objective of goals modeling is to define the possible ways, at high level, by which the services can be intelligently achieved. During the literature survey, the goals modeling in various domains is found (e.g. artificial intelligence, agents' goals or policy language) with their specific functions and concerns. This type of modeling is different from others due to a direct focus on the devices, their required states and possible ways by which the goals can achieved. Therefore, in this section, only high level modeling is covered. Moreover, some works related to the interaction modeling among goals, focusing on their conflicts, are found [148–150] but they are not entertained in this thesis due to our limited focus.

Our solution "Domotic Effects" provides the ability to the application designers to design the goals at higher level of abstraction, define their own operators for more complex goals and apply the required heuristics for selecting a suitable configuration among the set of possible configurations at runtime.

Katasonov [35] proposes high-level abstraction mechanism for easily managing the SmE by concentrating on the devices and software components at runtime. The abstractions are based on task (or sub-task). However the focus of this work seems limited due to author's disregard for the mechanism by which the organization of the task (their hierarchical nature) can be stored and achieve at run time. To counter and improve upon this identified weakness, our work not only provides the way how the tasks (goals) are stored but also goes one step ahead and illustrates how they can be achieved and enforced on the devices at run time.

Amigoni et al. designed a planning system, named as D-HTN [37], by involving the concepts of centralized and distributed planning from agent theory. For storing and using the list of actions associated with the respective goal, they used hierarchical task network (HTN) approach. The language "Task network" [151] on which HTN is based is static in

nature, and therefore does not allow the designers for defining new operators (which may vary from domain to domain). In this case, our solution capitalizes upon this limitation by not only modeling the goals in hierarchical structure, along with the sequences of actions, but also provides facilitates the designer for defining their own operators of interest.

Kaldeli et al. [39] propose an architecture for defining the goals in SmE. They also predefined the goals in declarative pattern, which is used by CSP (Constraint Satisfaction Problem) based planner for computing the plan (selection of configuration from the goal). In comparison with our solution, they do not facilitate the designers for defining their own operators according to their need for satisfying the requirements in intelligent way. Moreover, the planner is comparatively slower; it takes time in seconds whereas ours takes time in milliseconds for finding a solution.

Cheng et al. [33] propose a reasoning system for SmE, named as ASBR. By recording the history in form of scenarios, the system derives the preferences or habits of the residents and stores them in ontology. The apparent limitation of their work is their disregard for providing the storage mechanism; moreover the system is based upon the historical data and extensive coverage of all possible scenarios in the knowledge base can not be guaranteed. Our solution is capable of storing all possible configuration with the flexibility of imposing any type of constraints (e.g. energy optimization or context aware).

6.1.2 Evolution Finding

Our work aims at finding the evolution (path from source to destination) which is a combination of one or more individual transitions. There is a body of knowledge pertaining to similar generalized framework for evolution finding with procedural differences as part of their operation: state space search, path finding and test case generation are some of the methodologies in which the sequence of transitions from initial to destination states is extracted. On the other hand, target achievement is another foundation for a couple of methodologies in the body of knowledge which aims at finding the target state in which the device must be in order for a command to execute.

Rouillard and Tarby [152] present a solution to communicate with a home automation system using speech recognition. All actions that can be performed by residents are modeled with the help of a statechart (SCXML). The statechart consists of three basic entities, i.e. action, object and place. The action entity represents an action/operation requested by a user. The object entity represents a device on which the action is requested. The place entity indicates the location of the device. All entities are represented as concurrent substates in the statechart. When the user requests any operation, a message structure is filled with information regarding all three entities and the message is executed by the installed automation system. It is assumed that an action is equivalent to a command which may be used to perform the action. However, it may not be fully supportable in real smart spaces where not only the user intention may map to multiple devices (objects) but also, within each device, a sequence of commands may be needed to provide a specific utility.

Our work, on the other hand, takes into account the aforementioned issues by finding the evolution of each device and then enforcing it to achieve the user intentions.

For the automatic generation of test cases, it is important that the source state, the destination state and the correct sequence of events/actions are obtained from the statecharts [153–157]. Initially, it is required to flatten/normalize or expand the statecharts into a suitable format (such as Extended Finite State Machine (FSM), Kripke Strucure, Markov Chain, reachability tree or flowchart) according to the coverage criteria on the basis of variables and events (such as all-nodes, all-edges and all-paths) so that a detailed view can be obtained. Then, by using different FSMs or statechart based methods the test cases can be designed [157]. Compared to this, our work generates the abstracted action and state based behavioral state space graph using a model checker, which helps in finding the correct sequence of events/actions from any source to destination state.

In [155], Kansomkeat and Rivepiboon propose a technique for the automatic generation of test cases from statecharts. For this, statecharts are initially transformed into a Testing Flow Graph (TFG). On the basis of states, events (triggers), guards (conditions) and actions, an algorithm is designed through which the hierarchical structure and explicit control flow of the statecharts are transformed into a flattened TFG. Then, going from the root (initial) node of TFG to each leaf node, test sequences are designed which are further implemented for proving the correct behavior of the developed system. Compared to our work, this technique does not support the statechart parallelism concepts.

In [158], Hong et al. describe a method for the automatic generation of test cases from the specification of the system, modeled using statecharts. They adopt a model checking approach for the generation of test cases. Firstly, the semantics of statecharts using Kripke structures (a variant of automata used in model checking) are defined, then the statecharts are translated into the SMV model checker program. SMV is a CTL (State Based Branching time logic) [45] model checking approach, therefore the CTL temporal properties are designed by using the statecharts specifications; thus, the proposed control and data flow meet the coverage criteria. Each CTL formula contains one test case which returns true in the case of not matching with the specification; otherwise the model checker provides a counterexample, by following which a feasible (and executable) test case is designed with the observable events. Although both techniques have the advantages of model checking, compared to our work, this technique does not manage multiple states (as a single composite state or distinguished states) according to the feature value used for representing source and destination states, and it cannot find a compound transition through which such states can be reachable.

In [154], Hong et al. present their work for the selection of test sequences from statecharts. For this, the statecharts are initially normalized into Extended FSM format and then, by following the methods of Ural et al. [159], Extended FSMs are transformed into a flow graph which has the data and control flow of the system. The test sequences are designed by following all possible paths starting and ending with the initial state (configuration). Compared to our work, this technique does not support the filtering of

uninteresting events and features.

Stone et al. [147] describe an approach to support the modeling and validation of command sequences for space missions. They propose a *Checkable Sequence Language (CSL)* to model spacecraft components, sequences of commands and the assertions of flight rules. Given a sequence of commands, they validate whether any command violates the flight rules (represented as a future state) or not. For this purpose, they adopt a model checking approach. They try different sequences of commands and check whether or not the required destination state is reachable. In comparison, our work is capable of furnishing the correct sequence of commands from source to destination states instead of checking their suitability repeatedly. This, in turn, enhances the optimization of resources and implementation.

In the context of personalized remote/appliance control, the concepts of "Task-based button grouping" and "Macros" are proposed in the literature [160–162]. By using different techniques, such as machine learning or fuzzy logic, the set of commands (for accomplishing the task) are extracted and stored as user log files. In these concepts, the recorded data of a user command is used. Compared to our work, these techniques work with log files by maintaining a copy of the file for each user or machine. Moreover, these techniques can only address those destination states which are visited by the users/machines.

Due to the following collective advantages over the above mentioned techniques, our work is one step further and simpler:

- Most of the techniques spend more of their energy in defining the semantics to explicitly flatten/normalize the statechart of the system; whereas in our work, for obtaining the control and data flow graph of the system, a model checking technique (based upon theory of graph algorithms, data structures and logic) is used.

- The model checking technique (especially on-the-fly) has a capacity for covering and designing the exhaustive graph of the system.

- With the use of action abstraction, the unwanted events/actions can be ignored by model checking.

- The composition of states – not found in the above techniques – can be performed with the use of state abstractions.

- The sequence of minimum composite events/actions, through which the destination composite state from where the source is reachable (by only considering the interesting events/actions) can be obtained with the use of the abstracted graph generated by the model checking technique.

6.2 Problem Statement

A collective view of the states of all SmE devices at any point in time is referred as the *global state* $\mathcal{G}$ (of the environment). A change of state in any device has an effect on $\mathcal{G}$ and a new state $\mathcal{G}'$ will be produced. The goal g is associated with some devices and their corresponding destination states, and has an effect on $\mathcal{G}$. For achieving g, a *global evolution* $\mathcal{E}(g)$ is required by which $\mathcal{G}$ evolves to $\mathcal{G}'$, as expressed below:

$$\mathcal{G} \xrightarrow{\mathcal{E}(g)} \mathcal{G}'$$

The global evolution $\mathcal{E}(g)$ is a set of evolutions of each concerned device[1], $e(d_i)$, helping to realize the goal g. If m devices are involved, then the global evolution for satisfying g can be represented as below:

$$\mathcal{E}(g) = \{e(d_1), e(d_2) \ldots e(d_m)\}$$

The chapter addresses the problem of finding and enforcing a *global evolution* $\mathcal{E}$ which aims at satisfying the high-level goal g by evolving the entire environment from $\mathcal{G}$ to $\mathcal{G}'$.

6.3 TV Model: An Example

In the context of this chapter, it is assumed that the devices are independent: the working of one device will not depend/effect on the working of other devices in the environment. The interface modeling of the devices is performed by using the DogOnt ontology [23], whereas the behavior of each device (according to the information modeled in DogOnt) is textually encoded by using the semantics of State Chart Extensible Markup Language (SCXML) [145].

The TV model is used as a running example in the rest of this chapter. A tabular view of the interface modeling of TV in DogOnt is represented in Figure 6.2, and its behavioral modeling is represented in Figure 6.3. For the behavioral modeling of the devices, an assumption is made: the internal heuristics of the devices are ignored, such as *setChannel* and *setVolume* commands although modeled in DogOnt. As *setChannel* or *setVolume* commands directly move the TV to the required destination state by generating a single transition, these commands are excluded for the sake of detailed experimentations (finding more transitions in an evolution).

[1] An evolution for a single device may consist of more than one transition for evolving the device from a source state, ss, to a destination state, ds; $ss \xrightarrow{\{c_i[g_i]/a_i\}} s' \xrightarrow{\{c_{i+1}[g_{i+1}]/a_{i+1}\}} s''$ ... $\xrightarrow{\{c_n[g_n]/a_n\}} ds$. The sequence of these transitions, from a source state to a destination state, is known as an *evolution* e. For more details, consider Chapter 2 Section 2.1.2

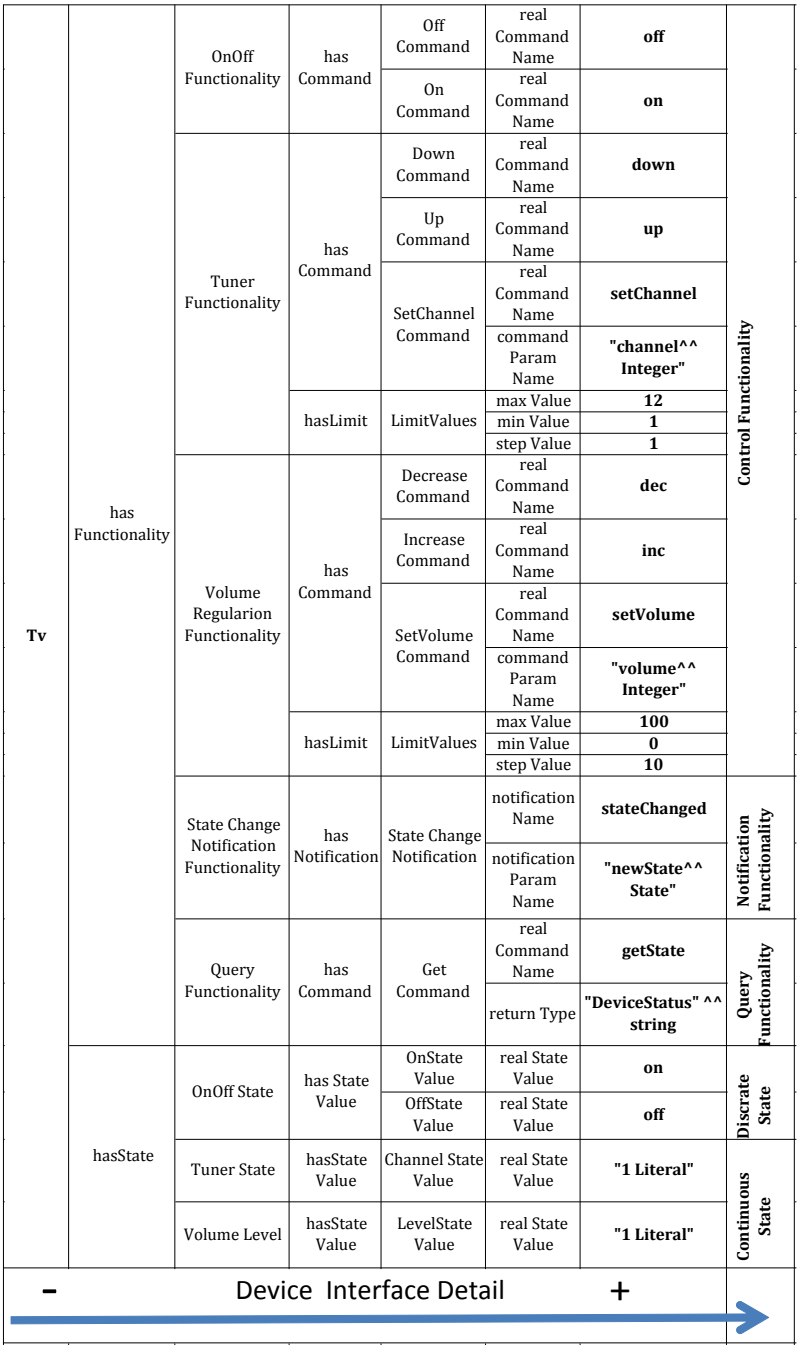

Figure 6.2. Interface Modeling of TV in DogOnt

In the behavioral modeling of a TV, *off* is considered as the initial state of TV, as represented in Figure 6.3. With the *on()* command, it moves from *off* to *on* state and, as

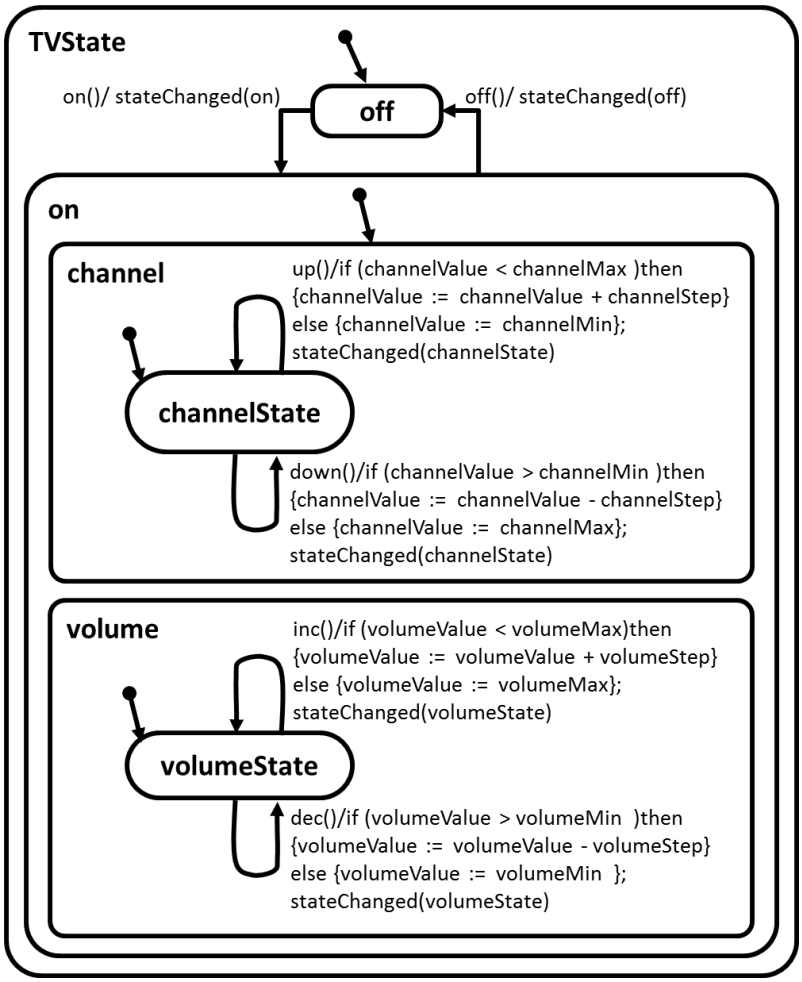

Figure 6.3. Behavioral Modeling of TV

a result, the TV will be on. The *on* state is a composite state in which the *channel* and *volume* (features of the TV) can be controlled. The initial value of channel is set to *6* and volume is set to *50%*. By using *up()* and *down()* commands, channel values can be adjusted with step *1*. The channel values are set in a loop and range from *1* to *12*. With the *up()* command at its maximum value, it wraps around and vice-versa with *down()* command. The *inc()* command is used for increasing the volume and *dec()* command for decreasing it, with a step of *10%*. The possible values of volume range from *0%* to *100%*. After completion the task, at each state, the device sends a *stateChanged(newState)* notification back as an acknowledgement about the status of task.

6.4 Goals Achievement Methodology

For finding and enforcing global evolution $\mathcal{E}(g)$, this chapter proposes a comprehensive methodology which attains its purpose in two steps: design-time and runtime. The framework of the proposed methodology is presented in Figure 6.4. At design-time, it generates the expanded behavioral graphs of all the devices with the help of a model checking technique. Then at runtime, the individual evolutions for the relevant devices are extracted by using these graphs and enforced with the help of different OSGi [56] services.

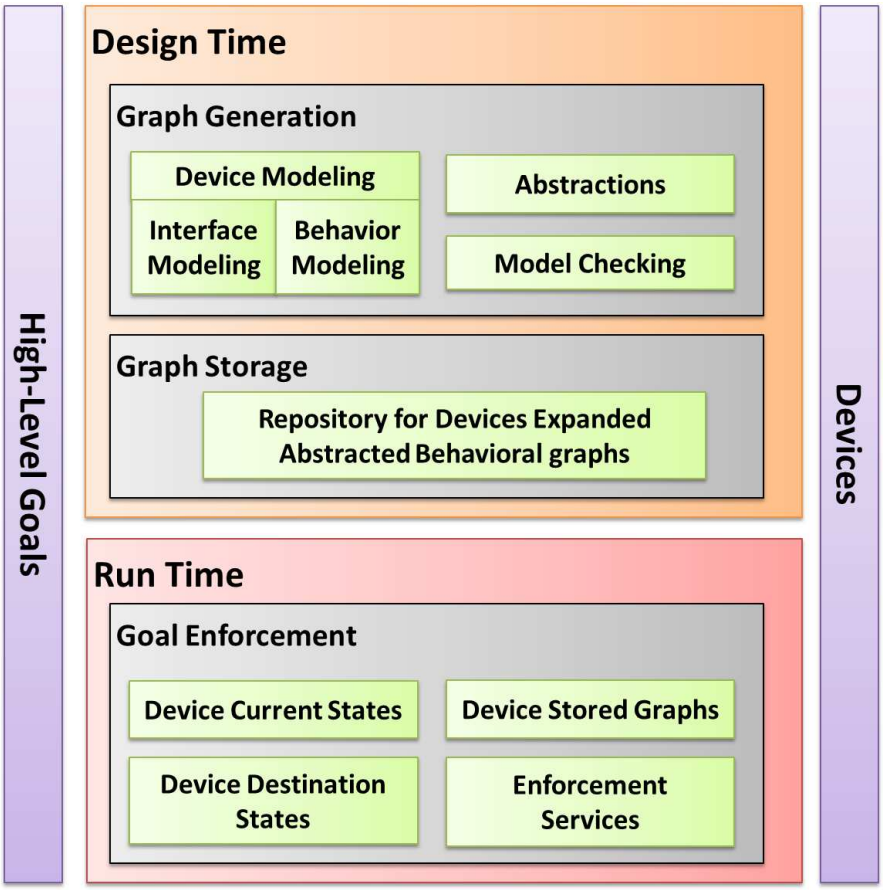

Figure 6.4. Framework of the Proposed Methodology

The following subsections explain the design-time and runtime methodologies. Each section contains a technique by which the methodology is implemented and an example is shown to demonstrate the proposed methodology.

6.4.1 Design-Time Methodology

The global evolution $\mathcal{E}(g)$ consists of a set of evolutions of various individual and independent devices $e(d_i)$. For finding the evolutions, their deterministic behavioral models are required which can behave like *real* devices in the *real* environment at design-time. As in the *real* environment, the commands can arrive in any order. Depending upon the current state, which is the manifestation of device configuration (a juxtaposition of feature and state values (defined in Section 2.1.1)), the commands which are acceptable at that particular state are considered and the rest are ignored (e.g. mute command, when the TV is in *off* state, is ignored). And when a device is performing any action against a command, and meantime another command arrives, depending upon the nature of the command and (device or system) specification, it can be immediately responded to (obstacle detected when the elevator door was in closing position), pushed into a stack/queue (elevator call request from the fourth floor when it is descending from second) or ignored/locked (elevator request from the same floor when it is already in the stack/queue).

A modeling framework is the work through which the devices can be synchronized with the environment by accepting the commands and sending the notifications back to the environment. Building upon the concept of such a closed environment from Chapter 5 Section 5.1.1, this chapter embeds the device in a synthetic environment as depicted in Figure 5.2. Similarly, the Environment Generates Commands (EGC) component is designed to send all possible commands to the device in any order. Based on the current state, which is a manifestation of device configuration (a juxtaposition of feature and state values (see Section 2.1.1) of Chapter 2), the relevant commands are accepted. Against the accepted command, the device model performs the relevant task and sends one (or more) notification message(s) which are received by the Environment Receives Notifications (ERN) component.

The current configuration of the device is updated with the change in any state or feature values, with which different transitions may precede the next configurations. The state-space of the system can be represented with a graph where all possible configurations are represented as different states (data flow by which the change in state or feature/-variable values can be identified) and the transitions by which the next configuration can be accessed are represented with edges (control flow). By traversing such graphs from one configuration to other, the complete sequence of transitions (labeled at edges) can be identified. This sequence of transitions is the required evolution. A graphical view of such an expanded behavioral graph is presented in Figure 6.5.

In the proposed framework, as the device modeling is kept enclosed in the environment, the expanded behavioral graph of the system (device interaction with the environment) exhibits the actual (data and control) flow of information when the device participates in the real environment. But the graph may contain more than one state with the same configuration. This is due to the collective nature of the graph in which the device model interacts with the environment model and the current configuration of the device

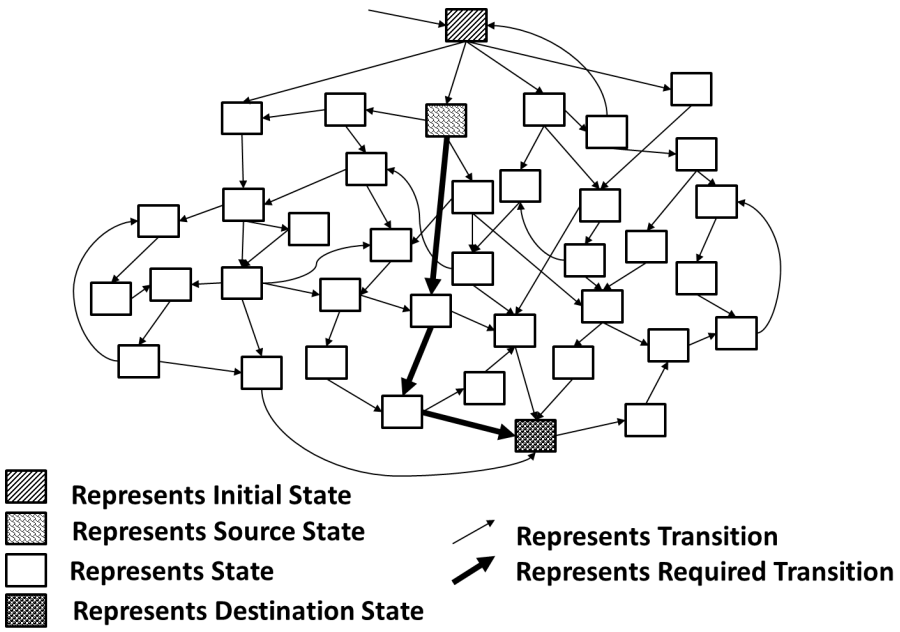

Figure 6.5. System Expansion

will reflect the states of EGC and ERN, and thus are annotated with these state as well.

At the time of traversing the graph, it is required to know the exact source and destination states so that the sequence of control flow can be identified. But due to the same configuration at more than one state, it becomes difficult to find the exact source and destination. For this, a solution is proposed: EGC sends the commands, some of which are accepted by the device based on its current configuration. Thus the edges on which the commands are accepted by the device model have been annotated (with the use of action/event abstraction). Similarly, the resultant configurations are also annotated (with the use of state abstraction). Since the command is initiated by the environment, the edge annotated with the command trailed by the annotated state (the annotated state which is before this annotated edge) is considered as the source state. After the completion of the task, the current configuration of the device is updated and one (or more) notification(s) are sent to the ERN. This, in return, discards the notification message and the discarding event is annotated on the edge. The annotated state trailed by the discarded message is considered as the destination state.

The model checking technique, due to the following advantages, provides the facility to generate such an expanded behavioral annotated data and control flow graph of the modeled system:

- It is strongly based on mathematics by underpinning the theory of graph algorithms, data structures and logic.

- Exhaustive coverage of all the possible scenarios, which a device/system may have to deal with, can be explored through it.

- It provides abstraction facilities by which the important concerns can be annotated with simple keywords and the rest of the information may remain hidden.

  – With the use of action/event abstraction, the more considerable commands/notifications can be annotated on the edges of the graph.

  – With the help of state abstraction, the resultant configurations can be annotated on the states of the system.

Design-Time Technique

For the implementation of the above design-time methodology, a technique is designed which is graphically represented in Figure 6.6 in which DogOnt (for the interface modeling of the devices), SCXML (for the behavioral modeling of devices) and UMC (a model checker used for generating the abstracted graph) are used as tools.

With the help of DogOnt, environment modeling for the concerned device is performed to automatically generate Environment Generate Commands (EGC) and Environment Receive Notifications (ERN) as components in the acceptable format of the model checker (UMC in our case). The EGC contains all the commands which a device can accept and ERN is capable of receiving all the notifications which a device can send. The model designer component converts the behavioral model of the device into the format of a model checker and concatenates it with the EGC and ERN. It also generates the instances of these components and stores the semi-completed model in a file.

The abstraction generator component takes the semi-completed model file, the list of (discrete type) states, the abbreviation of the offered features with their minimum, maximum, step values and the variable names which are used for representing these features in the behavioral modeling. Then it automatically generates all possible configurations based on the principle of Markov chains [163], in which permutations are performed starting from the list of states, the feature minimum to maximum values, along with their abbreviated characters (such as channel is abbreviated as C and volume is abbreviated as V) and increased by their step values.

Referring to the TV Model where the total number of discrete states $T(S)$ are *2* (*on* and *off*), and offered features $T(F)$ are *2* (channel and volume). The channel has total *12* possible values $T(V_1)$ (ranges from *1* to *12* with a step of *1*) and volume has *11* possible values $T(V_2)$ (ranges from *0* to *100* with a step of *10*). So, the total number of possible configurations which can be annotated with the use of state abstractions, $T(S_A)$, are calculated by the formula:

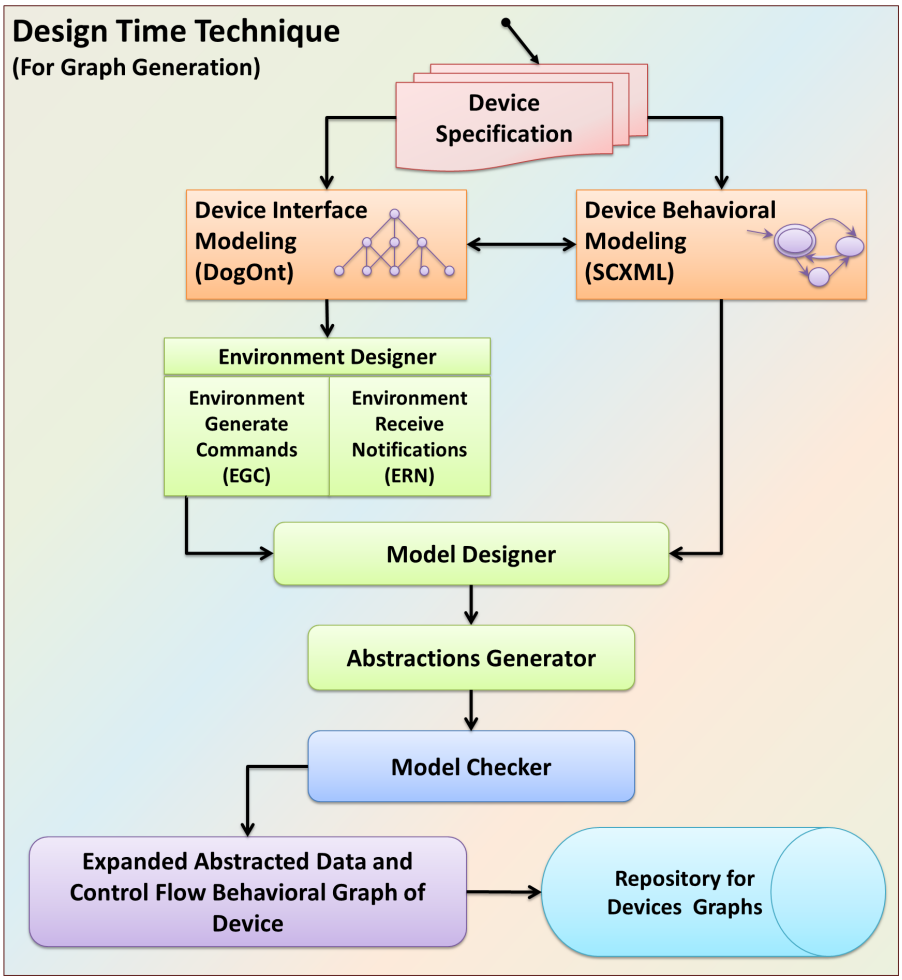

Figure 6.6. Methodology for Transition Finding

$$T(S_A) = T(S) \prod_{i=1}^{T(F)} T(V_i)$$

According to the formula ($T(S_A)$ = *2 (12 * 11)*), *264* possible configurations are generated by the abstraction generator. A configuration in which the device will be at *on* state, the channel value at *10* and volume value at *50*, is annotated as *onC10V50*, with the help of (*State inState(tv.on)and tv.channelValue=10 and tv.volumeValue=50 -> onC10V50*) state abstraction.

After the state abstraction, the abstraction generator component automatically generates both action abstractions (one is used for the identification of source state and the other is used for the identification of destination state) by incorporating the instance name of the devices (example is given in Figure 5.3). These state and action abstractions are added in the file. A complete file is known as a closed model, which is passed to the

model checker. The model checker has the power of generating the expanded abstracted data and control flow behavioral graph of this closed model.

For this purpose, UMC is used as a model checker tool which can generate the expanded abstracted data and control flow behavioral graph of the model. It also provide the facility to store the graph locally, for which it can export the graph in Scalable Vector Graphics (SVG) format. SVG is an open standard for designing the vector images in XML format, developed by the W3C. The images designed by SVG can easily be constructed, parsed and indexed. By this, the graph can be easily constructed and parsed for finding the source state, destination state and the evaluation by which the destination state can be reached.

The SVG graph of the closed model against the respective device name is stored in the repository for the further use at runtime.

Example

The closed model of the running TV example is represented in the UMC format in Figure 6.7[2]. A total number of *264* state abstractions are in this section, but due to limited space only two state abstractions with both action abstraction are represented in Figure 5.3. A fragment of the TV expanded abstracted data and control flow behavioral graph is presented in Figure 6.8.

The action abstraction *(Action tv.accept($1) -> $1)* is used for the identification of the source state. The $1 represents the name of the command which is accepted by the TV model (as *down*, *up* and *off* commands are annotated on the edges of Figure 6.8). The abstracted state trailed by the edge having command name is considered as the source state. Similarly, the *(Action $1:ec.return -> discardingReturn(ec))* is used for the identification of the destination state. The abstracted state trailed by the edge having *discardingReturn(ec)* annotation (as shown on the edges of Figure 6.8) is identified as the destination state (the internal logic for identification is presented in Section 6.4.1).

6.4.2 Runtime Methodology

The request for fulfilling a high-level goal g comes to the gateway. According to g, the gateway calls a function $F_{\mathcal{DS}}(g)$, in-result a configuration is obtained which consist of a list of the devices with their desired destination states required for satisfying g. Realizing this function will take the global state to $\mathcal{G}'$.

[2]It is divided into three sections: the first section has the statechart of the environment (EGC and ERN) components and the device; the second section (Objects) has the information of the instances of these statecharts; and the third section (Abstractions) has a list of abstractions.

```
Class State is
end State;
Class EGC is
 State top = E
 Transitions:
  E -> E {-/tv.down()}
  E -> E {-/tv.up()}
  E -> E {-/tv.off()}
  E -> E {-/tv.on()}
  E -> E {-/tv.inc()}
  E -> E {-/tv.dec()}
end EGC;
Class ERN is
 Operations: stateChanged(newState:State)
 State top = N
 Transitions:
  N -> N {stateChanged(newState)/}
end ERN;
Class TV is
 Operations: on(),off(),up(),down(),inc(),dec()
 Vars:channelValue:int=6, channelStep:int=1,channelMin:int=1,  channelMax:int=12,
      volumeValue:int=50, volumeStep:int=10,volumeMin:int=0 ,  volumeMax:int=100
 State TVState = off, on
 State on = channel/volume
 State channel = channelState
 State volume  = volumeState
 Transitions:
  off->on{on()/   notification.stateChanged(onState)}
  channelState->channelState{up()/
   if(channelValue<channelMax)then {channelValue:=channelValue+channelStep}
             else{channelValue:=channelMin};notification.stateChanged(channelState)}
  channelState->channelState{down()/
   if(channelValue>channelMin)then {channelValue:=channelValue-channelStep}
              else{channelValue:=channelMax}; notification.stateChanged(channelState)}
  volumeState->volumeState{inc()/
      if(volumeValue<volumeMax)then {volumeValue:=volumeValue+volumeStep}
              else{volumeValue:=volumeMax}; notification.stateChanged(volumeState)}
   volumeState->volumeState{dec()/
      if(volumeValue>volumeMin)then {volumeValue:=volumeValue-volumeStep}
              else{volumeValue:=volumeMin}; notification.stateChanged(volumeState)}
   on->off{off()/notification.stateChanged(offState)}
end TV
Objects:
   ec: EGC
   notification: ERN
   tv: TV
   channelState:State
   volumeState:State
   onState:State
   offState:State

Abstractions{
   Action tv.accept(\$1) -> \$1
   Action \$1:ec.return -> discardingReturn(ec)
   State inState(tv.on) and tv.channelValue=1 and tv.volumeValue=0 -> onC1V0
   State inState(tv.off)and tv.channelValue=7 and tv.volumeValue=40 -> offC7V40
   ...
}
```

Figure 6.7. Closed Model of TV in UMC format

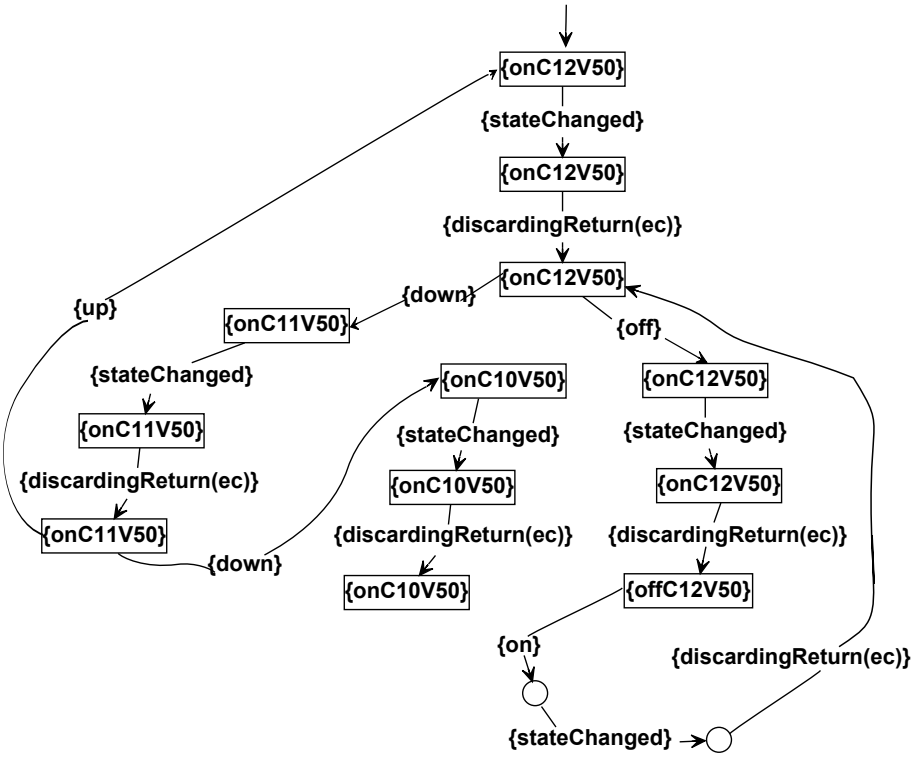

Figure 6.8. A Fragment of TV Graph

Let there be m devices involved for fulfilling g, then $F_{\mathcal{DS}}(g)$ returns to the gateway the list of all devices (specific instances of the device type) with the corresponding destination states ($ds(d_i), 1 \leq i \leq m$), as represented below:

$$F_{\mathcal{DS}}(g) \leftarrow \{ds(d_1), ds(d_2) \ldots ds(d_m)\}$$

Against the list of these devices returned from $F_{\mathcal{DS}}(g)$, the function $F_{\mathcal{SS}}(g)$ is activated by the gateway to attain the current states (globally representing $\mathcal{G}$) ($d_i(ss), 1 \leq i \leq m$) of these devices, as represented below:

$$F_{\mathcal{SS}}(g) \leftarrow \{ss(d_1), ss(d_2) \ldots ss(d_m)\}$$

On getting this list with the corresponding source and destination states ($d_i(ss, ds), 1 \leq i \leq m$), a function $F_{\mathcal{DT}}(g)$ will be called by the gateway, which returns the type of each device associated with g. Then, a function $F_e(g)$ will be modeled. The objective of the function is to arrange the source and the destination states along with the device identification, as represented below:

$F_{\mathcal{DT}}(g) \leftarrow \{dt(d_1), dt(d_2) \ldots dt(d_m)\}$

$F_e(g) = \{d_1(dt, ss, ds), d_2(dt, ss, ds) \ldots$
$d_m(dt, ss, ds)\}$

According to the $device\ type$ information returned by function $F_{\mathcal{DT}}(g)$, the function $F_e(g)$ firstly accesses the relevant expanded behavioral graphs of each device. Then the graph traversal algorithm is executed with the aim of identifying the source and destination states. The complete evolution against each device ($e(d_i), 1 \leq i \leq m$) is returned. The set of evolutions is referred to as the *global evolution* $\mathcal{E}(g)$ and is represented below:

$F_e(g) \leftarrow \{e(d_1), e(d_2) \ldots e(d_m)\}$

$\mathcal{E}(g) = \{e(d_1), e(d_2) \ldots e(d_m)\}$

Further, the *global evolution* $\mathcal{E}(g)$ is enforced on the relevant devices for fulfilling the request of selected goal g; as a result, the global configuration $\mathcal{G}$ of the environment will evolve to $\mathcal{G}'$, as represented below:

$\mathcal{G} \xrightarrow{\mathcal{E}(g)} \mathcal{G}'$

Runtime Technique

For the implementation of the runtime methodology, a technique is designed which is graphically represented in Figure 6.9.

In this technique, the high-level goal modeling is obtained by using the DogEffects ontology (see Section 2.1.3 of Chapter chap:Background) and device interface modeling is adopted from the DogOnt ontology [23]. A new module *Domotic Effects Executor* was built as an OSGi [56] bundle so that it can be integrated with the Domotic Effects framework [38, 40] and can communicate with the devices according to the desired requests/commands. The DE framework approach is generic in nature, but currently its implementation is built on top of Domotic OSGi Gateway (Dog) [55]. The expanded abstracted data and control flow behavioral graphs of the devices (which are in SVG format) are obtained and constructed at start-up of the SmE. The sequence and the steps performed for the implementation by using these tools are elucidated below:

1. A goal g can selected from the goals modeled in DogEffects ontology. Against which the enforcement request for the selected goals arrives at the Domotic OSGi Gateway (Dog), where the Domotic Effects (DE) framework is implemented.

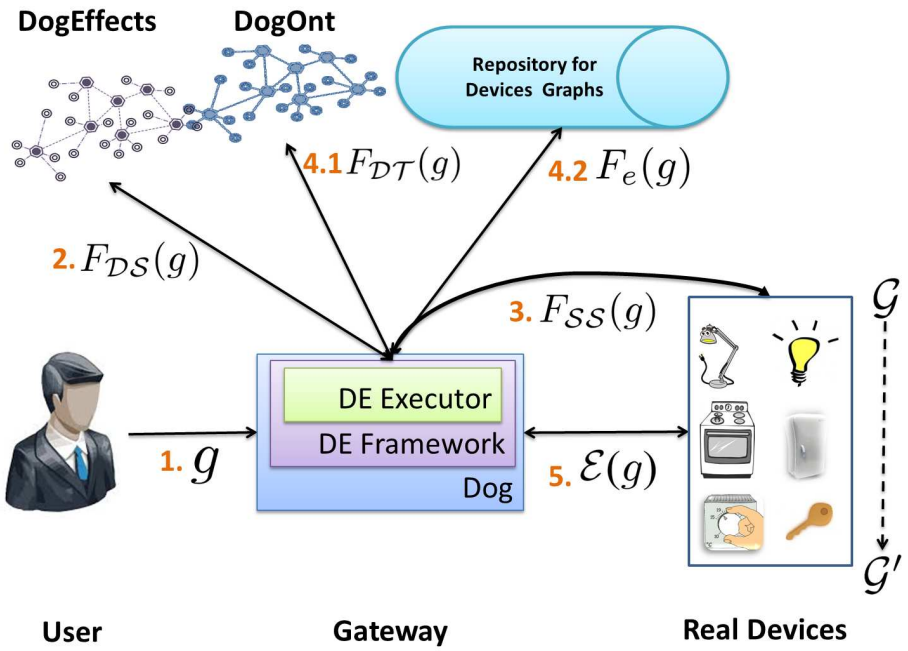

Figure 6.9. Steps of Domotic Effects Executor

2. The DE framework queries the DogEffects ontology and receives the complete tree along with the device names and their state, from the instance layer, associated with g (as presented in Fig. 2.2).

   This tree is transformed into Boolean satisfiability problem (SAT) [164]. The DE (SE or CE) are mapped as Boolean variables, and the expressions associated to the operator (defined in AmI layer) are firstly converted into the expression of basic logical operators (by the designed algorithms, mentioned in Section 2.1.3) and then mapped in terms of Boolean sub-expression. The heuristics/algorithms are designed for the expression associated to the operators which can not be converted into basic logical operators.

   These Boolean expressions are passed to the SAT solver. In our case, Sat4j solver is used, which is mature, open source, Java based and reasonably fast (as the results of our energy optimization work also show [38]) [165]. The SAT solver returns the possible configurations (devices with the desired destination states), then for the uncovered operators, their corresponding algorithms are run and the possible configurations are heuristically designed. Finally, a configuration is selected from them (the selection can be based on any constraints or factors of interest: energy optimization, context awareness or security demand). The selected configurations, against selected g, consists of a list of specific instances of the devices with their

required destination states.

3. According to the obtained list of device instances, their current state (source state) is queried by using the *DogState* bundle of Dog [55].

4. Thereafter, the device type of each instance is obtained by using the DogOnt ontology and accordingly its associated expanded behavioral graph is traversed with the help of the Djisktra shortest path (referred as evolution in this chapter) algorithm [166], which is a renowned algorithm for finding shortest path and compared to other algorithms, takes less time. Theoretically, the time complexity of the algorithm, when Fibonacci heap is used as a priority queue, is $O(|S| + |E| \log |E|)$ (in it S is total number of States and E is total number of edges) [167].

   The source and destination states are identified by using the logic presented in Section 6.4.1. The algorithm returns the shortest evolution from the source state to the destination against the instances of these device types.

5. The new *Domotic Effects Executor* module uses these evolutions and enforces them to the devices. These evolutions are enforced in parallel pattern.

After the enforcement of the global evolution, the configuration of the environment is updated. The new configuration is the required configuration which is associated with the goal.

In the case of selecting multiple goals at a time, the DE framework considers them as a single (complex) goal. In sequential pattern, it enforces them on the devices, and upon the successful enforcement of these goals, the required configuration of the environment will be obtained.

Example

The “Morning Wakeup” high-level goal is represented as an example. It is a combination of multiple individual activities: illuminating the bedroom, kitchen and bathroom; switching off the gas heater inside the bedroom; closing down the windows and raising their shutters; switching on the radio inside the bathroom; and switching on the TV inside the kitchen with volume configured to *70* and the channel to *8*.

As soon as the goal is selected, the appropriate request message was received at the gateway. The gateway queries the goal details from DogEffects which in turn returns the “Device Identification” and “Destination State” as mentioned in Table 6.1. The gateway further consults the identified devices to find out their source states as given in the column “Source State”. Later, the DE Executor queries the DogOnt for the device type from which it extracts the evolution path after traversing from the source state of the graph as given in column “Extracted Evolution”. A total of *17* devices participated to achieve the “Morning Wakeup” goal of which *8* were already in the required destination states and the

#	Device Identification	Source State	Destination State	Total Number of Commands	Total Number of Notifications	Extracted Evolution
1	SimpleLamp_lamp6_kitchen	offState	offState			
2	ShutterActuator_kitchen	upState	upState			
3	ShutterActuator_sh1_bedroom	upState	upState			
4	DoorActuator_d5_kitchen	closeState	closeState			
5	Radio_BathRoom	offState	onState	1	1	on, stateChanged
6	WindowActuator_w4_kitchen	openState	closeState	1	2	close, stateChanged, stateChanged
7	SimpleLamp_lamp2_bath	offState	onState	1	1	on, stateChanged
8	SimpleLamp_lamp9_bath	offState	onState	1	1	on, stateChanged
9	ShutterActuator_sh2_bedroom	upState	upState			
10	DoorActuator_d7_kitchen	closeState	closeState			
11	Tv_Kitchen	offC6V20	onC8V70	8	8	on, stateChanged, inc, stateChanged inc, stateChanged, inc, stateChanged up, stateChanged, inc, stateChanged inc, stateChanged, up, stateChanged
12	exhaustfan1	offState	onState	1	1	on, stateChanged
13	ShutterActuator_bath	upState	upState			
14	SimpleLamp_lamp8_bath	offState	onState	1	1	on, stateChanged
15	WindowActuator_w2_bedroom	openState	closeState	1	2	close, stateChanged, stateChanged
16	GasHeater_BedRoom	offState	offState			
17	WindowActuator_w1_bedroom	openState	closeState	1	2	close, stateChanged, stateChanged

Table 6.1. Morning Wakeup Goal Enforcement at Runtime

remaining *9* needed to follow the evolution. At successful completion of the individual activities, the high-level goal was fulfilled, the evolution was attained, and the global state was updated.

6.5 Experiment and Results

To prove the validity of the proposed methodology and measure different performance parameters, a set of experiments were carried out. In the research lab, we have some simple devices (switch, buttons, lamp and sensors) but for the experimentation purpose, the complex devices are also required, therefor in the absence of a real inhabited house, the emulation capabilities of the Dog gateway to simulate the behavior of devices was used. The complete house environment was simulated whose domotic structure was modeled as an instance of DogOnt ontology. The underlying domotic equipment may be handled by BTicino MyOpen, Z-Wave, ModBus and/or Knx.

In fact Dog simulates domotic environments thanks to the DogSim [168] library. A new test bundle was developed to test the validity of the approach and measure different performance parameters. Domotic effects, corresponding to generic goals (like securing or illuminating the house), were defined over the house.

The experiments ran on a standard personal computer with a quad-core Intel i5 processor and 4GB of RAM.

Tables 6.2 and 6.3 show the design-time experiment details, while Tables 6.4 and 6.5 show those of runtime.

The values under *Time to Construct a Graph (in Millisecond)* against each parameters of *Device Type* in Table 6.2 represent the time taken to access the files stored in repository in the SVG format and construct the graph of the devices (used in SmE). The *Total Number of States* and *Total Number of Edges* parameters represent the number of nodes and edges used to construct the graph. The *Total Number of Abstracted States* and *Total Number of Abstracted Edges* parameters show the number of abstracted nodes (which may occur more than once) and edges which are annotated according to the criteria defined in Section 6.4.1 (by the UMC model checker). The *Total Number of Possible Evolutions*, $T(E)$, are the maximum evolutions from any possible (selected abstracted) source state to any destination state of a device, and can be calculated by the given formula (from Section 6.4.1; $T(S_A)$ represents the total number of Abstracted states):

$$T(E) = T(S_A)^2 - T(S_A)$$

In the example of the TV model, there are *264* abstracted states therefore the possible evolutions for TV model according to the formula are *69432*. The average time taken for computing all possible evolutions $T(E)$ against each device are given under the parameter *Average Time to compute all Possible Evolutions (in Millisecond)*. Given the

Device Type	Time to Construct a Graph (in Millisecond)	Total Number of States	Total Number of Abstracted States	Total Number of Edges	Total Number of Abstracted Edges	Total Number of Possible Evolutions	Average Time (in Millisecond) to compute all Possible Evolutions
Dimmer Lamp	681.47	48	12	70	58	132	1.6
Door Actuator	659.28	16	4	17	13	12	2.89
Shutter Actuator	712.69	20	5	24	19	20	1.92
Simple Lamp	586.36	8	2	8	6	2	0.69
TV	3455.04	1320	792	1848	1584	69432	7.66
Window Actuator	740.57	20	5	23	18	20	1.84

Table 6.2. Device Type Graph Contents

Device Type	Source State	Destination State	Length of Path	Number of Commands	Number of Notifications	Time (in Millisecond) for each Evolution
Dimmer Lamp	on50	on100	30	5	5	2.53
	offState	on80	24	4	4	1.77
Door Actuator	closeState	openState	12	1	2	0.69
	closingState	openingState	6	1	1	2.00
Shutter Actuator	downState	upState	12	1	2	1.07
	upState	HalfOpenState	12	2	2	1.08
Simple Lamp	offState	onState	6	1	1	0.72
	onState	offState	6	1	1	0.50
TV	onC8V60	offC12V0	66	11	11	6.27
	onC10V20	onC9V100	54	9	9	3.10
	onC9V100	onC7V40	48	8	8	3.09
	onC12V80	onC1V90	12	2	2	0.82
Window Actuator	openState	closeState	12	1	2	0.86
	closingState	openingState	12	1	2	1.14

Table 6.3. Evolution Construction Details from Each Device Type (Some Samples)

space constraints, the detailed representation of $T(E)$ is not possible, therefore a sample set of few evolutions is presented in Table 6.3. The *Length of Path* parameter shows the total number of (abstracted or unabstracted) states, by passing which the *Destination State* will be reachable from the corresponding *Source State*. The respective evolution consists of commands and notifications, the *Number of Commands* and *Number of Notifications* parameters show their total number (along with their sequences). The *Time for each evolution (in Millisecond)* represents the time of each evolution which is taken (by the Djisktra shortest path algorithm) to identify it.

High-Level Goal Number	Secure Home	Home Illumination	Morning Wakeup	Air Passage	Afternoon Lunch	Bath Illumination
1	✗	✗	✗	✓	✗	✗
2	✗	✗	✓	✗	✗	✗
3	✗	✓	✗	✗	✗	✗
4	✗	✗	✓	✗	✗	✓
5	✗	✗	✗	✓	✗	✓
6	✗	✓	✗	✗	✗	✓
7	✗	✓	✗	✗	✓	✓
8	✓	✗	✗	✗	✗	✗
9	✓	✓	✗	✗	✗	✗
10	✓	✓	✗	✗	✓	✗
11	✓	✗	✗	✗	✗	✓
12	✓	✓	✗	✗	✗	✓
13	✓	✓	✗	✗	✓	✓

Table 6.4. Selected High-Level Goals for 6 Use Cases

For enforcing high-level goals on the SmE at runtime, a random selection of *13* use cases is made from the possible combinations of *6* use cases ($2^6 = 64$, or *63* if the trivial case is omitted, where no DE is enforced on the environment). These are *Secure Home*, *Home Illumination*, *Morning Wakeup*, *Air Passage*, *Afternoon Lunch* and *Bath Illumination*.

The “Secure Home” use case secures all the exit points of the house, i.e. all exit doors and windows. This use case consists of several DEs providing the ability to secure different rooms of the house. It can be used in case of emergency, theft and robbery.

The “Home Illumination” use case requires that all the rooms of the house be illuminated. Illumination can either be natural or artificial. For natural illumination window shutters (depending upon outside illumination) can be opened. For artificial illumination lights and lamps can be switched on.

The “Morning Wakeup” use case represents a typical scenario when a resident wants

High-level Goal Number	Total Number of Devices	Total Number of Active Devices	Total Number of Devices in Destination State	Total Number of Commands	Total Number of Notifications	Enforcement Time (in Millisecond)
1	15	9	6	18	18	95.40
2	17	11	6	19	19	87.01
3	24	9	15	12	12	58.64
4	17	3	14	5	5	38.53
5	19	8	11	15	15	66.66
6	24	6	18	11	11	48.55
7	27	6	21	9	9	63.14
8	20	8	12	19	19	25.47
9	28	2	26	2	2	29.52
10	31	3	28	6	6	79.38
11	23	3	20	5	5	48.50
12	28	5	23	8	8	51.75
13	31	0	31	0	0	44.42

Table 6.5. Device Activation Statistics

to perform a sequence of activities after waking up in morning, like illuminating the bedroom, the kitchen and the bathroom, switching off the gas heater inside the bedroom, and switching on the kitchen television and the bathroom radio (More details and formulation of each use cases are given in [40].

The “Air Passage” use case regulates the passage of natural air inside the home by controlling the windows and their shutters.

The “Afternoon Lunch” use case heats the oven, closes the kitchen door to prevent the cooking odor from entering inside the home and switches on the kitchen TV. The “Bathroom Illumination” use case lights up the bathroom using natural or artificial lights based on time of the day and external conditions.

These use cases are tabulated in Table 6.4 in such a way that the first goal considers the home in its “initial” state (as a source state), where all the appliances are switched off. Moving on, every preceding goal (destination state) treats the previous one as the source state. Also, every goal is a complex combination of previously described six user cases: tick (✓) denotes the use case as “selected” and the cross (✗) as “not selected” (which does not necessarily mean “inactive”).

For the selected goals, Table 6.5 shows the number of devices involved in enforcing the DE under the parameter *Total number of Devices* against the reference number of the selected goal (mentioned in Table 6.4). In the selected goal, chances are that some devices are already in their required destination states and some are required to evolve for reaching there. The value under *Total Number of Active Devices* represents the number of devices which are required to evolve, whereas *Total Number of Devices in Destination State* mentions the number of devices which are already in the specified destination state. For the devices which should evolve, the number of commands to be executed (*Total Number of Commands*) and the number of notification received (*Total Number of Notifications*) for enforcing the selected goals are given under these parameters. The time taken for enforcing the overall selected goal is given under the parameter *Enforcement Time (in Millisecond)*.

A thorough analysis of Table 6.2 shows that the TV device type has the maximum number of states and accordingly the times to construct the graph and compute all possible evolutions are the maximum among all the tested device types. The reason for a larger time span for computing all possible evolution is attributed to the high number of states and the longer length of paths due to continuity of long range values. Whereas the other device types, which mostly offer limited functionality, have lower number of states and accordingly lower time requirements.

Further, analyzing Table 6.3, it can be observed that TV device type takes the maximum time for single evolution. As mentioned in the previous description as well, the TV has a lengthy path with a higher number of commands and notifications. This results in consumption of more time. Whereas Door Actuator, Shutter Actuator, Simple Lamp and Wind Actuator, with only 1 command to execute and shorter paths, take less time for individual evolutions.

Moving on, the analysis of Table 6.5 shows that the enforcement time is highly dependent on the number of devices to be activated. For consequent goals, where one (or many) devices have already reached their desired state, the enforcement time is considerably lower. But in cases where a majority of devices are required to switch their states, the enforcement time is understandably higher. Also, it is imperative to consider the sequence of operations; for the consequent goals, if the device activation is already done by the previous goal, the enforcement operation is usually light.

6.6 Discussion

The experiments allow the evaluation of the overall methodology proposed here. The time consumption in the case of complex devices is observed to be higher owing to the higher number of states, commands, notifications and the continuous type of long ranging values of these devices. Simple devices, on the other hand, demonstrate a rather light and insubstantial time and processing load. Also the sequence of operation plays important role in minimizing the time as the methodology is designed to avoid any redundancy of operations.

The integration with Dog and the results prove the applicability of the proposed methodology to real systems. In addition, it shows that the approach satisfies the requirement of performing most operations in real time. Notwithstanding the strengths of this research, it disregards the possible conflicts arising in goals due to complex scenarios. Further, a typical user is not necessarily over-incentivized due to lack of goal personalization with the help of user-friendly GUI environment. Thus, the future research aims at resolving the potential conflicts and deadlocks of user goals in complex scenarios and providing user-friendly graphical environment where users may take the liberty of personalizing the system according to his/her needs and desires.

Chapter 7

Discussion and Conclusion

Smart Environments (SmE) are a growing field which provides implicit computation facilities in the environment so that they behave in a sophisticated and desired manner. This sophistication is achieved with the interaction of users with the sensors, actuators, electrical appliances and hidden computation. The versatile nature of these components and their interaction render the systems huge, complex and ambiguous, motivating to use the formal verification for validating the desired behavior. As formal methods, especially model checking, have various advantages over other techniques, which are (also listed in the thesis) given below:

- they are strongly based on mathematical evidence and increase the understandability of the modeled system;
- they are underpinning the theory of graph algorithms, data structures and logic;
- they are used for reliably modeling a system at design time;
- they can explore the exhaustive coverage of all the possible scenarios, which a device/system model may have to deal with;
- they can model the concerning requirements in the form of properties by using logic based on mathematics;
- they can formally verify the modeled system against the requirements (reliable behavior, along with other requirements of the system);
- they provides abstraction facilities by which the important concerns can be annotated with simple keywords and the rest of the information may remain hidden.
  - With the use of action/event abstraction, the more considerable commands/notifications can be annotated on the edges of the graph.

  - With the help of state abstraction, the resultant configurations can be annotated on the states of the system.

- they can trace back the errors and can help in fixing them at early design stages.

Formal verification of SmE (and its related components) is performed by various researchers but it is found that there is a sizable research gap in SmE modeling and verification area. Mostly complex modeling and verification scenarios and components are given less or no attention partly due to the inherent complexity and partly due to personal inclination of current researchers towards areas of their interest. This hinders in providing holistic solutions leaving behind the industry and users with their specific needs and demands.

For this, an in-depth survey of existing literature and state-of-the art techniques is performed. In which, the techniques which are used for the modeling of SmE and its related components, along with the conformance of reliable behavior through formal verification approaches, are considered.

These techniques are analyzed by empirically driving some parameters related to the focused area, modeling formalism, formal verification and other important factors. The analysis conclude that the techniques mostly follow Statecharts for the modeling purpose. It was also observed that the black box modeling, owing to lack of its visibility, is scarcely diffused in the techniques. Nevertheless, it assumes a fundamental role by providing generic dictionaries and naming/communication conventions which help broadly at the time of implementation. It is also observed that very few techniques model and verify – at a deeper level – all basic components of SmE (user, devices, control algorithm, environment/context). The model checking technique is used for the formal verification. Some techniques also use abstractions for reducing the state-space of the model. Results of the survey show that no technique is fully automatic in true nature and covers all the dimensions (e.g. modeling of context, user, devices) of SmE. Therefore, it is deduced that more R&D effort, impartial and objective in its nature, needs to be put into the SmE modeling and verification research.

A comprehensive methodology is required which may entail all the major components, for the design and verification, of SmE; users, devices, environment and control algorithms. On the basis of the analysis of the literature survey and the empirically deduced parameters, a methodology is proposed. The proposed methodology consists of ten steps, starting from the requirement organization of each component, with the elaboration of important aspects, to the verification of entire SmE. The organized specifications provide a better understandability of the system through which the ambiguities (during modeling and verification) can be sufficiently reduced. Further, the probability of missing any properties has been efficiently controlled by requirement organization.

It is attempted to bridge the discovered research gap from the literature survey in the proposed methodology. By keeping in view the covered and uncovered modeling areas

of other surveyed methodologies/techniques, their focused concerns, and the verification aspects, a methodology is proposed. In the proposed methodology, it is tried to integrate the advantages, covered and uncovered areas, of survey techniques with the use of mostly adopted and richer semantic tools.

The proposed methodologies use both type of formalisms for the modeling purpose: black box and white box. The black box modeling is performed with the use of Ontology where white box modeling is performed by using the semantics of Harel statechart. The intelligence is modeled with the use of event-condition-action strategy. The desired behavior/aspects which are required to verify, on the modelÂ¸ is formatted in properties by using the syntax and semantic of Action-and-State based temporal logic, known as UCTL. For the modeling of SmE all basic components: Users devices, control strategy, context and their interaction are considered. All the behavioral aspects of the users are considered except the action history and their division on the basis of roles are not considered. The interaction among various components of SmE, at various levels, is considered in the proposed methodology. The verificational aspects related to different components and levels are considered except the real time and probabilistic verification (as such type of modeling is not performed).

The methodology is implemented through a set of designed techniques (individual device verification and entire system verification) and implemented on (small but not so simple) real life systems. Almost 80% work, including the conversion of device behavioral models into the accepted format of model checker, the generation of temporal properties related to verify the consistency, the generation of environment component, abstractions, instances, the integration of these models into the acceptable format of model checker, and saving the complete model in a file are automatically performed. The rest of the task, the interface and behavioral modeling of devices, the modeling of control strategy, the designing of complex temporal properties related to behavioral verification and checking of these properties on the model are manually performed. Moreover, with the use of action and state abstraction the unnecessary details are kept hidden; in result the only limitation of model checking technique is sufficiently reduced.

The first run of verification processes did not achieve all the properties as satisfactory against the models. After appropriate modifications to the models, it was then proven to conform to the organized requirements. The successful results demonstrate the consistency, correctness, reliability, safety, security and desired behavior of the modeled SmE system and its related modeled components: user behavior modeling, device modeling, environment/context modeling, control algorithm modeling and their interaction. These behavioral verified models can be used safely, for any purpose, in various design and implementation phases. A closer look of the proposed methodology with the reference of surveyed techniques are presented in Table 7.1 and 7.2.

More advanced SmE requirements, related to High Level description of various user goals, are also achieved by automatically activating the devices with the use of their verified behavioral models. As, the goals are associated with a multitude of resources and

Researchers	Black Box Modeling	White Box Modeling	Intelligence Modeling	Requirements Modeling	Users Modeling	Devices Modeling	Control Modeling	Context Modeling	Interaction Modeling
Ahmed and Tripathi [68]	✘	Role based collaboration model	Role based	LTL	UPr, UA		✔		UI, IC, CO
Augusto and Hornos [69]	✘	Activity Modeling Through Promela processes	Event (Activity detection), Condition(location identification), Action (operation graded)	LTL	UI, UP, UA, UB		✔	✔	US, UC, SC, CO
Augusto and McCullagh [10]	✘	Finite State Machine	Event Condition Action	TCTL	UA	Behavior	✔		US, UI, SC, IC, CO
Benghazi et al. [70]	✘	UML-RT , CSP+T	Event Condition (previous history) Action	F_{TT} (Common Formal Semantic Domain)	UH, UA		✔		US, UI, SC, IC, CO
Bernardeschi et al. [13]	✘	CCS/MEIJE Process Algebra	Event Condition Action	mu-ACTL			✔		IC, CO
Bonhomme et al. [21]	System Engineering Standards, EIA-632	Petri-Nets, HiLes	Decision Logic	Temporal Properties	UI, UH, UA		✔		US, UI, SC, IC, CO
Boytsov and Zaslavsky [71]	Context Space Theory (CST)	Orthotope-based Situation Space	Weighted Rule Based	Situation Algebra Expression			✔		IC
Corno and Sanaullah [58, 113, 169]	**Ontology**	**Statecharts**	**Event Condition Action**	**UCTL**	**UI, UP, UA, UB**	**Behavior**	✔	✔	**US, UC, UI, SC, IC, CO**
Coronato and Pietro [15, 75, 76]	Ontology	Ambient Calculus	Ambient movement, Pre-and-Post conditions	Ambient logic + RTTL	UI, UP, UB		✔	✔	US, UC, SC, CO
Gnesi et al. [77]	✘	Hierarchical Statecharts	Event Condition Action	ACTL	UA	Behavior			UI
Gnesi and Mazzanti [79]	✘	Communicating State Machines	Event Condition Action	mu-ACTL	UA, UB	Behavior		✔	UC, UI
Hoogendoorn et al. [80, 81]	✘	Predicate logic	Rule Based	TTL	UH, UA		✔		UI, IC, CO
Ishikawa et al. [86]	✘	Event Calculus	Rule Based	Axioms Based through Discrete Event Calculus	UI, UPr, UP, UA		✔	✔	US, UC, UI, SC, CO
Leelaprute et al. [19]	Object Oriented Modeling, System description	Object Oriented Modeling, Service description	Event Condition Action	CTL		Behavior	✔		IC, CO
Liu et al. [89]	✘	CSP#	Rule Based	LTL	UI, UA		✔	✔	US, UC, UI, SC, IC, CO
[90–92]	✘	PVS Logic, a Typed higher-ordered Logic	✘	Axioms Based (according to property template)	UI, UPr, UA	Behavior			UI
Ranganathan and Campbell [93]	✘	Ambient Calculus	Rule Based, DL-Based, Relational Algebra	Ambient Logic	UI, UA		✔	✔	US, UC, UI, SC, IC, CO

Table 7.1. Formal Modeling analysis with the Proposed Methodology

Researchers	**Consistency Verification**	Entire System Verification						**Abstraction**	**Automatic**	**Scalability**	**Verification Tool**
		Users Behavior Verification	Context Verification	Device Behavior Verification	Devices Interaction Control Verification	Real Time Verification	Probabilistic Verification				
Ahmed and Tripathi [68]		✔			✔			✔	Automatic		SPIN
Augusto and Hornos [69]		✔	✔		✔			✘	Manual	✔	SPIN
Augusto and McCullagh [10]				✔	✔	✔		✘	Manually	✔	UPPAL
Benghazi et al. [70]	✔				✔	✔		✘	Semi-automatic	✔	✘
Bernardeschi et al. [13]					✔			✔	Manually		JACK
Bonhomme et al. [21]	✔(Behavioral Analysis)				✔			✘	Semi-automatic		TINA
Boytsov and Zaslavsky [71]	✔							✘	Manual		Algorithms
Corno and Sanaullah [58, 113, 169]	✔	✔	✔	✔	✔			✔	**Semi-automatic**	✔	**UMC**
Coronato and Pietro [15, 75, 76]		✔	✔			✔		✘	Semi-automatic	✔	Ambient Designer, Nu-SMV
Gnesi et al. [77]				✔				✔	Manually		JACK
Gnesi and Mazzanti [79]		✔	✔					✔	Manually	✔	UMC
Hoogendoorn et al. [80, 81]					✔	✔		✘	Semi-automatic		TTL Checker, SMV
Ishikawa et al. [86]		✔	✔		✔	✔		✔	Manual		Discrete Event Calculus Reasoner
Leelaprute et al. [19]				✔	✔			✔	Semi-automatic		SMV
Liu et al. [89]			✔		✔		✔	✘	Semi-automatic	✔	PAT
[90–92]	✔			✔				✘	Semi-automatic		PVS
Ranganathan and Campbell [93]		✔	✔		✔			✘	Manually		specified in [119–121]

Table 7.2. Formal Verification analysis with the Proposed Methodology

can relate to acquiring the functionalities of a single device or a group of devices (e.g. sensors, actuators, lamps or TV). The variety of devices has raised a major problem of managing SmE. An increasingly adopted solution to the problem is the modeling of goals and intentions, and then using artificial intelligence to control the respective SmE accordingly. Generally, the solution advocates that the goals can be achieved by controlling the evolution of the states of the devices. In order to reach a particular state, an automatic device activation methodology is proposed, which uses the verified behavioral models of the concerning devices and considers *a*) the composite nature of the state of an individual device; *b*) the possible variation of specific commands, notifications and their sequence based on the current states of the devices.

The methodology works at two levels: design-time and runtime. At design-time, it constructs a data and control flow behavioral graph of the verified device models, based upon the concepts of a model checking approach. Then at runtime, on the arrival of any request for the enforcement of a goal, it accordingly consults these graphs and extracts a reliable shortest evolution for all the devices which have to be affected by the desired goal. Then, these extracted evolutions are enforced on the corresponding devices and, as a result, the desired high-level goal will be automatically accomplished.

A detailed experimentation is conducted which shows that the time consumption in the case of complex devices is observed to be higher owing to the higher number of states, commands, notifications and the continuous type of long ranging values of these devices. Simple devices, on the other hand, demonstrate a rather light and insubstantial time and processing load. Also the sequence of operation plays important role in minimizing the time as the methodology is designed to avoid any redundancy of operations. In addition, it shows that the approach satisfies the requirement of performing most operations in real time. Notwithstanding the strengths of the research, it disregards the possible conflicts arising in High Level goals due to complex scenarios. Further, a typical user is not necessarily over-incentivized due to lack of goal personalization with the help of user-friendly GUI environment. Thus, the future research aims at resolving the potential conflicts and deadlocks of user goals in complex scenarios and providing user-friendly graphical environment where users may take the liberty of personalizing the system according to his/her needs and desires.

Bibliography

[1] M. Weiser. The Computer for the 21st Century. *Scientific American*, 265(3):94–104, 1991.

[2] D. Cook and S. Das. *Smart environments: Technology, protocols and applications*, volume 43. Wiley-Interscience, 2004.

[3] D.J. Cook, J.C. Augusto, and V.R. Jakkula. Ambient intelligence: Technologies, applications, and opportunities. *Pervasive and Mobile Computing*, 5(4):277–298, 2009.

[4] F. Sadri. Ambient Intelligence: A Survey. *ACM Computing Surveys*, 43(4):36:1–36:66, October 2011.

[5] E. Aarts. Ambient intelligence: Basic elements and insights. *it-Information Technology*, 50(1):7–12, 2008.

[6] D. Saha and A. Mukherjee. Pervasive computing: a paradigm for the 21st century. *Computer*, 36(3):25–31, 2003.

[7] G. Acampora and V. Loia. Fuzzy Control Interoperability and Scalability for Adaptive Domotic Framework. *IEEE Transactions on Industrial Informatics*, 1(2):97–111, 2005.

[8] D.J. Cook. Multi-agent smart environments. *Journal of Ambient Intelligence and Smart Environments*, 1(1):51–55, 2009.

[9] C. Diane and D. Sajal. *Smart Environments: Technology, Protocols and Applications*. Wiley-Interscience, 2004.

[10] J.C. Augusto and P. Mccullagh. Ambient Intelligence: Concepts and Applications. *Computer Science and Information Systems*, 4 (1):1–27, 2007.

[11] A. Steventon and S. Wright. *Intelligent spaces: the application of pervasive ICT*. Springer-Verlag New York Inc, 2006.

[12] K. Ducatel, M. Bogdanowicz, F. Scapolo, J. Leijten, and J.C. Burgelman. Scenarios for Ambient Intelligence in 2010. Technical report, ISTAG: IST Advisory Group, February 2001.

[13] C. Bernardeschi, A. Fantechi, S. Gnesi, S. Larosa, G. Mongardi, and D. Romano. A Formal Verification Environment for Railway Signaling System Design. *Formal Methods in System Design*, 12(2):139–161, 1998.

[14] E.M. Clarke and J.M. Wing. Formal Methods: State of the Art and Future Directions. *ACM Computing Surveys*, 28(4):626–643, 1996.

[15] A. Coronato and G. De Pietro. Formal Design of Ambient Intelligence Applications. *Computer*, 43(12):60 –68, December 2010.

[16] F. Wang. Formal Verification of Timed Systems: A Survey and Perspective. *Proceedings of the IEEE*, 92(8):1283–1305, 2004.

[17] A. Gupta. Formal hardware verification methods: A survey. *Formal Methods in System Design*, 1(2):151–238, 1992.

[18] D. Bonino and F. Corno. Domains: Domain-based modeling for ambient intelligence. *Pervasive and Mobile Computing*, 8(4):614–628, 2012.

[19] P. Leelaprute, M. Nakamura, T. Tsuchiya, K. Matsumoto, and T. Kikuno. Describing and Verifying Integrated Services of Home Network Systems. In *12th Asia-Pacific Software Engineering Conference*, page 10, December 2005.

[20] D. Muthiayen. Animation and formal verification of real-time reactive systems in an object-oriented environment. A Thesis in the Department of Computer Science, 1996.

[21] S. Bonhomme, E. Campo, D. Esteve, and J. Guennec. Methodology and tools for the design and verification of a smart management system for home comfort. In *Intelligent Systems, 2008. IS'08. 4th International IEEE Conference*, volume 3, pages 24–2. IEEE, 2008.

[22] M. R. Kakoee, H. Shojaei, H. Ghasemzadeh, M. Sirjani, and Z. Navabi. A new approach for design and verification of transaction level models. In *Circuits and Systems, 2007. ISCAS 2007. IEEE International Symposium on*, pages 3760 –3763, may 2007.

[23] D. Bonino and F. Corno. DogOnt - Ontology Modeling for Intelligent Domotic Environments. In Amith Sheth, Steffen Staab, Mike Dean, Massimo Paolucci, Diana Maynard, Timothy Finin, and Krishnaprasad Thirunarayan, editors, *International Semantic Web Conference*, number 5318 in LNCS, pages 790–803. Springer-Verlag, October 2008.

[24] E. Meshkova, J. Riihijarvi, P. Mahonen, and C. Kavadias. Modeling the home environment using ontology with applications in software configuration management. In *Telecommunications, 2008. ICT 2008. International Conference on*, pages 1–6. IEEE, 2008.

[25] J. Rouillard, X. Le Pallec, J.C. Tarby, and R. Marvie. Facilitating the Design of Multi-channel Interfaces for Ambient Computing. In *2010 Third International Conference on Advances in Computer-Human Interactions*, pages 95–100. IEEE, 2010.

[26] M.H. Coen. Design principles for intelligent environments. In *Proceedings of the fifteenth national/tenth conference on Artificial intelligence/Innovative applications of artificial intelligence*, AAAI '98/IAAI '98, pages 547–554, Menlo Park, CA, USA, 1998. American Association for Artificial Intelligence.

[27] H. Chen, F. Perich, T. Finin, and A. Joshi. Soupa: standard ontology for ubiquitous and pervasive applications. In *Mobile and Ubiquitous Systems: Networking and Services, 2004. MOBIQUITOUS 2004. The First Annual International Conference on*, pages 258 – 267. IEEE Computer Society, aug. 2004.

[28] H. Chen, T. Finin, and A. Joshi. A context broker for building smart meeting rooms. In *Proceedings of the Knowledge Representation and Ontology for Autonomous Systems Symposium*. Defense Technical Information Center, 2004.

[29] L. Sommaruga, A. Perri, and F. Furfari. Domoml-env: an ontology for human home interaction. In *Proceedings of SWAP*, pages 14–16. Citeseer, 2005.

[30] T.R. Gruber et al. A translation approach to portable ontology specifications. *Knowledge acquisition*, 5:199–199, 1993.

[31] D. Fensel. *Ontologies: a silver bullet for knowledge management and electronic commerce*. Springer-Verlag, New York, NY, USA, 2001.

[32] D. Harel. Statecharts: A visual formalism for complex systems. *Science of computer programming*, 8(3):231–274, 1987.

[33] S.T. Cheng, C.H. Wang, and C.C. Chen. An adaptive scenario based reasoning system cross smart houses. In *Communications and Information Technology, 2009. ISCIT 2009. 9th International Symposium on*, pages 549–554. IEEE, 2009.

[34] H. Dibowski, J. Ploennigs, and K. Kabitzsch. Automated design of building automation systems. *Industrial Electronics, IEEE Transactions on*, 57(11):3606–3613, 2010.

[35] A. Katasonov. Enabling non-programmers to develop smart environment applications. In *Computers and Communications (ISCC), 2010 IEEE Symposium on*, pages 1059–1064. IEEE, 2010.

[36] P. Rashidi and D.J. Cook. Keeping the resident in the loop: Adapting the smart home to the user. *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, 39(5):949 –959, 9 2009.

[37] F. Amigoni, N. Gatti, C. Pinciroli, and M. Roveri. What planner for ambient intelligence applications? *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, 35(1):7 – 21, 1 2005.

[38] F. Corno and F. Razzak. Intelligent energy optimization for user intelligible goals in smart home environments. *Smart Grid, IEEE Transactions on*, 3(4):2128–2135, 2012.

[39] E. Kaldeli, E. Warriach, J. Bresser, A. Lazovik, and M. Aiello. Interoperation, composition and simulation of services at home. *Service-Oriented Computing*, pages 167–181, 2010.

[40] F. Corno and F. Razzak. Sat based enforcement of domotic effects in smart environments. *Journal of Ambient Intelligence and Humanized Computing*, pages 1–15, 2013.

[41] D.L. McGuinness, F.V. Harmelen, et al. Owl web ontology language overview. *W3C recommendation*, 10(2004-03):10, 2004.

[42] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems*. Springer-Verlag New York, Inc., New York, NY, USA, 1992.

[43] Franco Mazzanti. *UMC 3.3 User Guide, ISTI Technical Report 2006-TR-33*. ISTI-CNR Pisa-Italy, September 2006.

[44] R. De Nicola and F. Vaandrager. Action versus state based logics for transition systems. *Semantics of Systems of Concurrent Processes, Lecture Notes in Computer Science*, 469:407–419, 1990.

[45] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic Specifications. *ACM Transactions on Programming Languages and Systems*, 8:2:244–263, April 1986.

[46] M.H. Ter Beek, A. Fantechi, S. Gnesi, and F. Mazzanti. An action/state-based model-checking approach for the analysis of communication protocols for service-oriented applications. In *Proceedings of the 12th international conference on Formal methods for industrial critical systems*, pages 133–148. Springer-Verlag, 2007.

[47] M.H. Ter Beek, S. Gnesi, N. Koch, and F. Mazzanti. Formal verification of an automotive scenario in service-oriented computing. In *Software Engineering, 2008. ICSE'08. ACM/IEEE 30th International Conference on*, pages 613–622. IEEE, 2008.

[48] Matthew Hennessy and Robin Milner. On observing nondeterminism and concurrency. In Jaco de Bakker and Jan van Leeuwen, editors, *Automata, Languages and Programming*, volume 85 of *Lecture Notes in Computer Science*, pages 299–309. Springer Berlin / Heidelberg, 1980.

[49] C. Baier and J.P. Katoen. *Principles of model-checking*. The MIT Press, ISBN-10: 0-262-02649-X ISBN-13:978-0-262-02649-9, May 2008.

[50] P. Cousot and R. Cousot. Abstract interpretation frameworks. *Journal of logic and computation*, 2(4):511, 1992.

[51] O. Wei. *Abstraction for verification and refutation in model checking*. PhD thesis, University of Toronto, 2009.

[52] S. Gnesi and F. Mazzanti. A Model Checking Verification Environments for UML Statecharts. In *Proceedings of the XLIII Congresso Annuale AICA*, 2005.

[53] F. Mazzanti. Designing UML models with UMC. Technical report, Technical Report 2009-TR-43, ISTI-CNR-Pisa, Italy, 2009.

[54] R.D. Nicola. Three Logics for Branching Bisimulation. *Journal of the Association for Computing Machinery*, 42:2:458–487, March 1995.

[55] D. Bonino, E. Castellina, and F. Corno. The DOG gateway: Enabling Ontology-based Intelligent Domotic Environments. *IEEE Transactions on Consumer Electronics*, 54(4):1656 –1664, November 2008.

[56] OSGi Alliance. Osgi service platform, core specification, release 4, version 4.1. *OSGi Specification*, 2007.

[57] D. Marples and P. Kriens. The open services gateway initiative: An introductory overview. *Communications Magazine, IEEE*, 39(12):110–114, 2001.

[58] F. Corno and M. Sanaullah. Formal verification of device state chart models. In *Intelligent Environments (IE), 2011 7th International Conference on*, pages 66 –73, july 2011.

[59] Lars Birkedal, Søren Debois, Ebbe Elsborg, Thomas Hildebrandt, and Henning Niss. Bigraphical models of context-aware systems. In *Foundations of software science and computation structures*, pages 187–201. Springer, 2006.

[60] Cecilia Mascolo, Dan Ghica, Mark Ryan, and Emil Lupu. UbiVal: Fundamental Approaches to Validation of Ubiquitous Computing Applications and Infrastructures. Research Proposed EP/D076625/2, EPSRC, 2009.

[61] Francois Siewe, Hussein Zedan, and Antonio Cau. The Calculus of Context-Aware Ambients. *Journal of Computer and System Sciences*, 77(4):597–620, 2011.

[62] Gruia-Catalin Roman, Christine Julien, and Jamie Payton. Modeling Adaptive Behaviors in Context UNITY. *Theoretical Computer Science*, 376(3):185–204, 2007.

[63] Vojtěch Forejt, Marta Kwiatkowska, Gethin Norman, and David Parker. Automated Verification Techniques for Probabilistic Systems. In *Formal Methods for Eternal Networked Software Systems*, pages 53–113. Springer, 2011.

[64] Luca Mottola, Thiemo Voigt, Fredrik Österlind, Joakim Eriksson, Luciano Baresi, and Carlo Ghezzi. Anquiro: Enabling Efficient Static Verification of Sensor Network Software. In *Proceedings of the ICSE Workshop on Software Engineering for Sensor Network Applications*, pages 32–37. ACM, 2010.

[65] Peng Li and John Regehr. T-check: Bug Finding for Sensor Networks. In *Proceedings of the 9th ACM/IEEE International Conference on Information Processing in Sensor Networks*, pages 174–185. ACM, 2010.

[66] Matthew L. Bolton, Ellen J. Bass, and Radu I. Siminiceanu. Generating Phenotypical Erroneous Human Behavior to Evaluate Human-Automation Interaction using Model Checking. *International Journal of Human-Computer Studies*, 70(11):888 – 906, 2012.

[67] Jit Biswas, Mounir Mokhtari, Jin Song Dong, and Philip Yap. Mild Dementia Care at Home–Integrating Activity Monitoring, User Interface Plasticity and Scenario Verification. In *Aging Friendly Technology for Health and Independence*, pages 160–170. Springer, 2010.

[68] Tanvir Ahmed and Anand R. Tripathi. Static Verification of Security Requirements in Role Based CSCW Systems. In *Proceedings of the Eighth ACM Symposium on Access Control Models and Technologies*, SACMAT '03, pages 196–203, New York, NY, USA, 2003. ACM.

[69] J.C. Augusto and M. J. Hornos. Software Simulation and Verification to Increase the Reliability of Intelligent Environments. *Advances in Engineering Software*, 58:18–34, 2013.

[70] K. Benghazi, M.V. Hurtado, M.J. Hornos, M.L. Rodríguez, C. Rodríguez-Domínguez, A.B. Pelegrina, and M.J. Rodríguez-Fórtiz. Enabling Correct Design

and Formal Analysis of Ambient Assisted Living systems. *Journal of Systems and Software*, 85(3):498–510, 2012.

[71] A. Boytsov and A. Zaslavsky. Formal Verification of Context and Situation Models in Pervasive Computing. *Pervasive and Mobile Computing*, 9(1):98 – 117, 2013.

[72] Juan Ye, Simon Dobson, and Susan McKeever. Situation Identification Techniques in Pervasive Computing: A Review. *Pervasive and Mobile Computing*, 8(1):36–66, 2012.

[73] Amir Padovitz, Seng W Loke, and Arkady Zaslavsky. Multiple-Agent Perspectives in Reasoning about Situations for Context-Aware Pervasive Computing Systems. *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, 38(4):729–742, 2008.

[74] Lotfi Asker Zadeh. Fuzzy Sets. *Information and control*, 8(3):338–353, 1965.

[75] A. Coronato and G.D.E. Pietro. Formal Specification of Wireless and Pervasive Healthcare Applications. *ACM Transactions on Embedded Computing Systems*, 10(1):12, 2010.

[76] A. Coronato and G. De Pietro. Formal Specification and Verification of Ubiquitous and Pervasive Systems. *ACM Transactions on Autonomous and Adaptive Systems*, 6(1):9:1–9:6, February 2011.

[77] S. Gnesi, D. Latella, and M. Massink. Model Checking UML Statechart Diagrams Using JACK. In *Proceedings of 4th IEEE International Symposium on High-Assurance Systems Engineering*, pages 46–55. IEEE, 1999.

[78] D. Latella, I. Majzik, and M. Massink. Towards a Formal Operational Semantics of UML Statechart Diagrams. In *Proceedings of the IFIP TC6/WG6*, volume 99, pages 15–18, 1999.

[79] S. Gnesi and F. Mazzanti. On the Fly Model Checking of Communicating UML State Machines. In *Second ACIS International Conference on Software Engineering Research, Management and Applications*, pages 331–338, 2004.

[80] Mark Hoogendoorn, MichelC.A. Klein, Zulfiqar A. Memon, and Jan Treur. Formal Verification of an Agent-Based Support System for Medicine Intake. In Ana Fred, Joaquim Filipe, and Hugo Gamboa, editors, *Biomedical Engineering Systems and Technologies*, volume 25 of *Communications in Computer and Information Science*, pages 453–466. Springer Berlin Heidelberg, 2009.

[81] Mark Hoogendoorn, Michel C.A. Klein, Zulfiqar A. Memon, and Jan Treur. Formal Specification and Analysis of Intelligent Agents for Model-Based Medicine Usage Management. *Computers in Biology and Medicine*, 43(5):444 – 457, 2013.

[82] T. Bosse, C.M. Jonker, L.V.D. Meij, A. Sharpanskykh, and J. Treur. Specification and Verification of Dynamics in Agent Models. *International Journal of Cooperative Information Systems*, 18(01):167–193, 2009.

[83] T. Bosse, C. M. Jonker, L.V.D. Meij, and J. Treur. A Language and Environment for Analysis of Dynamics by Simulation. *International Journal on Artificial Intelligence Tools*, 16(03):435–464, 2007.

[84] A.A. Aziz, M.C.A. Klein, and J. Treur. An Integrative Ambient Agent Model for Unipolar Depression Relapse Prevention. *Journal of Ambient Intelligence and Smart Environments*, 2(1):5–20, 2010.

[85] Alexei Sharpanskykh and Jan Treur. An Ambient Agent Architecture Exploiting Automated Cognitive Analysis. *Journal of Ambient Intelligence and Humanized Computing*, 3(3):219–237, 2012.

[86] Fuyuki Ishikawa, Basem Suleiman, Kayoko Yamamoto, and Shinichi Honiden. Physical Interaction in Pervasive Computing: Formal Modeling, Analysis and Verification. In *Proceedings of the international conference on Pervasive services*, pages 133–140. ACM, 2009.

[87] Murray Shanahan. The Event Calculus Explained. In *Artificial intelligence today*, pages 409–430. Springer, 1999.

[88] IBM. Commonsense Reasoning with the Discrete Event Calculus Reasoner, 2005.

[89] Y. Liu, X. Zhang, J.S. Dong, Y. Liu, J. Sun, J. Biswas, and M. Mokhtari. Formal Analysis of Pervasive Computing Systems. In *17th International Conference on Engineering of Complex Computer Systems*, pages 169–178. IEEE, 2012.

[90] Paolo Masci, Dominic Furniss, Paul Curzon, Michael D Harrison, and Ann Blandford. Supporting field investigators with pvs: a case study in the healthcare domain. In *Software Engineering for Resilient Systems*, pages 150–164. Springer, 2012.

[91] Paolo Masci, Paul Curzon, Michael D Harrison, Anaheed Ayoub, Insup Lee, and Harold Thimbleby. Verification of interactive software for medical devices: Pca infusion pumps and fda regulation as an example. *EICS2013. ACM Digital Library*, 2013.

[92] Paolo Masci, Yi Zhang, Paul Curzon, Michael D Harrison, Paul Jones, and Harold Thimbleby. Verification of software for medical device user interfaces in PVS. Submitted paper, School of Electronic Engineering and Computer Science, Queen Mary University of London, United Kingdom, 2013.

[93] A. Ranganathan and R.H. Campbell. Provably Correct Pervasive Computing Environments. In *Sixth Annual International Conference on Pervasive Computing and Communications*, pages 160–169. IEEE, 2008.

[94] Witold Charatonik and Jean-Marc Talbot. The Decidability of Model Checking Mobile Ambients. In Laurent Fribourg, editor, *Computer Science Logic*, volume 2142 of *Lecture Notes in Computer Science*, pages 339–354. Springer Berlin / Heidelberg, 2001.

[95] G. Booch, J. Rumbaugh, and I. Jacobson. *Unified Modeling Language User Guide, The*. Addison Wesley., October 1998.

[96] D. Fensel. *Ontologies: A Silver Bullet for Knowledge Management and Electronic Commerce*. Springer-Verlag, New York, NY, USA, 2001.

[97] J.C.M. Baeten. A Brief History of Process Algebra. *Theoretical Computer Science*, 335(2-3):131–146, 2005.

[98] J.A. Bergstra and J.W. Klop. Process Algebra for Synchronous Communication. *Information and control*, 60(1-3):109–137, 1984.

[99] M. Nielsen, G. Plotkin, and G. Winskel. Petri nets, event structures and domains, part i. *Theoretical Computer Science*, 13(1):85–108, 1981.

[100] J.E. Hopcroft, R. Motwani, and J.D. Ullman. *Introduction to Automata Theory, Languages, and Computation*, volume 2. Addison-wesley Reading, MA, 1979.

[101] Erich Mikk, Yassine Lakhnechi, and Michael Siegel. Hierarchical automata as model for statecharts. In R.K. Shyamasundar and K. Ueda, editors, *Advances in Computing Science– ASIAN'97*, volume 1345 of *Lecture Notes in Computer Science*, pages 181–196. Springer Berlin Heidelberg, 1997.

[102] D.N. Jansen, H. Hermanns, and J.P. Katoen. A Probabilistic Extension of UML Statecharts. In *Formal Techniques in Real-Time and Fault-Tolerant Systems*, pages 355–374. Springer, 2002.

[103] M. Hennessy and R. Milner. Algebraic Laws for Nondeterminism and Concurrency. *Journal of the ACM*, 32(1):137–161, 1985.

[104] C.A.R. Hoare. Communicating Sequential Processes. *Communications of the ACM*, 21(8):666–677, 1978.

[105] S. Brookes. On the Relationship of CCS and CSP. *Automata, Languages and Programming*, pages 83–96, 1983.

[106] R. Milner, J. Parrow, and D. Walker. A Calculus of Mobile processes, I. *Information and computation*, 100(1):1–40, 1992.

[107] L. Cardelli and A. Gordon. Mobile Ambients. In *Foundations of Software Science and Computation Structures*, pages 140–155. Springer, 1998.

[108] H. Hagras, V. Callaghan, M. Colley, G. Clarke, A. Pounds-Cornish, and H. Duman. Creating an Ambient-Intelligence Environment Using Embedded Agents. *Intelligent Systems*, 19(6):12–20, 2004.

[109] Witold Pedrycz. Human Centricity in Computing with Fuzzy Sets: an Interpretability Quest for Higher Order Granular Constructs. *Journal of Ambient Intelligence and Humanized Computing*, 1(1):65–74, 2010.

[110] V. Stankovski and J. Trnkoczy. Application of Decision Trees to Smart Homes. *Designing Smart Homes*, pages 132–145, 2006.

[111] D. Cook, M. Youngblood, and S. Das. A Multi-Agent Approach to Controlling a Smart Environment. *Designing smart homes*, pages 165–182, 2006.

[112] A. Kofod-Petersen and A. Aamodt. Contextualised Ambient Intelligence Through Case-Based Reasoning. *Advances in Case-Based Reasoning*, pages 211–225, 2006.

[113] F. Corno and M. Sanaullah. Design time Methodology for the Formal Verification of Intelligent Domotic Environments. In *International Symposium on Ambient Intelligence*. Springer, April 2011.

[114] L. Feng, P. Apers, and W. Jonker. Towards Context-Aware Data Management for

Ambient Intelligence. In *Database and Expert Systems Applications*, pages 422–431. Springer, 2004.

[115] M.H.T. Beek, A. Fantechi, S. Gnesi, and F. Mazzanti. A State/Event-Based Model-Checking Approach for the Analysis of Abstract System Properties. *Science of Computer Programming*, 76:119–135, February 2011.

[116] A. Fantechi, S. Gnesi, A. Lapadula, F. Mazzanti, R. Pugliese, and F. Tiezzi. A model checking Approach for Verifying COWS Specifications. *Fundamental Approaches to Software Engineering*, pages 230–245, 2008.

[117] R. Alur and T. Henzinger. Logics and Models of Real Time: A Survey. In *Real-Time: Theory in Practice*, pages 74–106. Springer, 1992.

[118] M. Reynolds. An Axiomatization of PCTL*. *Information and Computation*, 201(1):72–119, 2005.

[119] D. Latella, I. Majzik, and M. Massink. Automatic verification of a behavioural subset of uml statechart diagrams using the spin model-checker. *Formal Aspects of Computing*, 11(6):637–664, 1999.

[120] M.D.M. Gallardo, P. Merino, and E. Pimentel. Debugging UML Designs with Model Checking. *Journal of Object Technology*, 1(2):101–117, 2002.

[121] E. Mikk, Y. Lakhnech, M. Siegel, and G.J. Holzmann. Implementing Statecharts in PROMELA/SPIN. In *Proceedings in 2nd Workshop on Industrial Strength Formal Specification Techniques*, pages 90 –101. IEEE, 1998.

[122] E. Clarke, O. Grumberg, and D. Long. Verification Tools for Finite-State Concurrent Systems. *A Decade of Concurrency Reflections and Perspectives*, pages 124–175, 1994.

[123] T. Schafer, A. Knapp, and S. Merz. Model Checking UML State Machines and Collaborations. *Electronic Notes in Theoretical Computer Science*, 55(3):357–369, 2001.

[124] J. Lilius and I.P. Paltor. vUML: A Tool for Verifying UML Models. In *14th IEEE International Conference on Automated Software Engineering*, pages 255–258. IEEE, 1999.

[125] G.J. Holzmann. The Model Checker SPIN. *IEEE Transactions on Software Engineering*, 23(5):279–295, 1997.

[126] K.L. McMillan. Symbolic Model Checking: An Approach to the State Explosion Problem. Technical report, DTIC Document, 1992.

[127] M.H. ter. Beek, F. Mazzanti, and S. Gnesi. Cmc-umc: a framework for the verification of abstract service-oriented properties. In *Proceedings of the 2009 ACM symposium on Applied Computing*, pages 2111–2117, New York, NY, USA, 2009.

[128] K.G. Larsen, P. Pettersson, and W. Yi. Uppaal in a nutshell. *International Journal on Software Tools for Technology Transfer*, 1(1):134–152, 1997.

[129] A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella. Nusmv 2: An Opensource Tool for Symbolic Model Checking. In *Computer Aided Verification*, pages 241–268. Springer, 2002.

[130] B. Berthomieu and F.Vernadat. Time Petri Nets Analysis with TINA. In *Third International Conference on Quantitative Evaluation of Systems*, pages 123–124. IEEE, 2006.

[131] J. Byg, K. Jørgensen, and J. Srba. TAPAAL: Editor, Simulator and Verifier of Timed-arc Petri Nets. *Automated Technology for Verification and Analysis*, pages 84–89, 2009.

[132] G. Gardey, D. Lime, M. Magnin, and O. Roux. Romeo: A tool for Analyzing Time Petri Nets. In *Computer Aided Verification*, pages 261–272. Springer, 2005.

[133] G. Madl, S. Abdelwahed, and D.C. Schmidt. Verifying Distributed Real-Time Properties of Embedded Systems via Graph Transformations and Model Checking. *Real-Time Systems*, 33(1):77–100, 2006.

[134] P. Stevens and C. Stirling. Practical Model-Checking Using Games. *Tools and Algorithms for the Construction and Analysis of Systems*, pages 85–101, 1998.

[135] H. Garavel, F. Lang, R. Mateescu, et al. An Overview of CADP 2001. Research Report RT-0254, INRIA, 2001.

[136] Jun Sun, Yang Liu, JinSong Dong, and Jun Pang. PAT: Towards Flexible Verification under Fairness. In Ahmed Bouajjani and Oded Maler, editors, *Computer Aided Verification*, volume 5643 of *Lecture Notes in Computer Science*, pages 709–714. Springer Berlin Heidelberg, 2009.

[137] M. Kwiatkowska, G. Norman, and D. Parker. PRISM: Probabilistic Symbolic Model Checker. *Computer Performance Evaluation: Modelling Techniques and Tools*, pages 113–140, 2002.

[138] J. Harrison. HOL Light: A Tutorial Introduction. In *Formal Methods in Computer-Aided Design*, pages 265–269. Springer, 1996.

[139] B. Barras, S. Boutin, C. Cornes, J. Courant, J.C. Filliatre, E. Gimenez, H. Herbelin, G. Huet, C. Munoz, C. Murthy, et al. *The Coq Proof Assistant Reference Manual: Version 6.1*. INRIA– Institut National De Recharche En Informatique Et Automatique, May 1997.

[140] B. Brock, M. Kaufmann, and J. Moore. ACL2 Theorems about Commercial Microprocessors. In *Formal Methods in Computer-Aided Design*, pages 275–293. Springer, 1996.

[141] Sam Owre, Sreeranga Rajan, John M Rushby, Natarajan Shankar, and Mandayam Srivas. Pvs: Combining specification, proof checking, and model checking. In *Computer Aided Verification*, pages 411–414. Springer, 1996.

[142] Lawrence C Paulson. The Foundation of a Generic Theorem Prover. *Journal of Automated Reasoning*, 5(3):363–397, 1989.

[143] D. Crocker. Perfect Developer: A Tool for Object-Oriented Formal Specification and Refinement. *Tools exhibition notes at formal methods Europe*, 2003.

[144] D. Crocker and J. Carlton. Verification of C Programs using Automated Reasoning. In *Fifth International Conference on Software Engineering and Formal Methods*, pages 7–14. IEEE, 2007.

[145] J. Barnett, R. Akolkar, R.J. Auburn, et al. State chart xml (scxml): State machine notation for control abstraction. *W3C Working Draft*, 2007.

[146] L. Ryzhyk, P. Chubb, I. Kuz, E. Le Sueur, and G. Heiser. Automatic device driver synthesis with Termite. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, pages 73–86. ACM, 2009.

[147] C.A. Stone, A. Carter, H.L. Justice, R.M. Keller, and Y.W. Tung. Improved modeling and validation of command sequences using a checkable sequence language. In *Aerospace Conference, 2012 IEEE*, pages 1–11. IEEE, march 2012.

[148] T. Heider and T. Kirste. Supporting goal-based interaction with dynamic intelligent environments. In *ECAI*, pages 596–602. Fraunhofer Publica (Germany), 2002.

[149] J.L. Encarnaçao and T. Kirste. Ambient intelligence: Towards smart appliance ensembles. *From Integrated Publication and Information Systems to Information and Knowledge Environments*, pages 261–270, 2005.

[150] M. Hellenschmidt. Distributed implementation of a self-organizing decentralized multimedia appliance middleware. In N. Davies, T. Kirste, and H. Schumann, editors, *Mobile Computing and Ambient Intelligence: The Challenge of Multimedia*, number 05181 in Dagstuhl Seminar Proceedings, Dagstuhl, Germany, 2005. IBFI, Germany.

[151] K. Erol, J. Hendler, and D.S. Nau. Complexity results for HTN planning. *Annals of Mathematics and Artificial Intelligence*, 18(1):69–93, 1996.

[152] J. Rouillard and J.C. Tarby. How to communicate smartly with your house? *International Journal of Ad Hoc and Ubiquitous Computing*, 7(3):155–162, 2011.

[153] K. Bogdanov and M. Holcombe. Statechart testing method for aircraft control systems. *Software testing, verification and reliability*, 11(1):39–54, 2001.

[154] H.S. Hong, Y.G. Kim, S.D. Cha, D.H. Bae, and H. Ural. A test sequence selection method for statecharts. *Software Testing, Verification & Reliability*, 10(4):203–227, 2000.

[155] S. Kansomkeat and W. Rivepiboon. Automated-generating test case using uml statechart diagrams. In *Proceedings of the 2003 annual research conference of the South African institute of computer scientists and information technologists on Enablement through technology*, pages 296–300. South African Institute for Computer Scientists and Information Technologists, 2003.

[156] Z. Pap, I. Majzik, and A. Pataricza. Checking general safety criteria on UML statecharts. *Computer Safety, Reliability and Security*, pages 46–55, 2001.

[157] V. Santiago, N.L. Vijaykumar, D. Guimarães, A.S. Amaral, and E. Ferreira. An environment for automated test case generation from statechart-based and finite state machine-based behavioral models. In *Software Testing Verification and Validation Workshop, 2008. ICSTW'08. IEEE International Conference on*, pages 63–72. IEEE, 2008.

[158] H.S. Hong, I. Lee, and O. Sokolsky. Automatic test generation from statecharts using model checking. *In FATES '01.BRICS Notes Series*, NS-01-4(4):1–21, 2001.

[159] H Ural, K Saleh, and A Williams. Test generation based on control and data dependencies within system specifications in sdl. *Computer Communications*, 23(7):609–627, 2000.

[160] C.L. Isbell Jr, O. Omojokun, and J.S. Pierce. From devices to tasks: automatic task prediction for personalized appliance control. *Personal and Ubiquitous Computing*, 8(3-4):146–153, 2004.

[161] O. Omojokun, C. Isbell, and P. Dewan. Towards automatic personalization of device controls. *IEEE Transactions on Consumer Electronics*, 55(1):269–276, 2009.

[162] O. Omojokun, S. Pierce, L. Isbell, and P. Dewan. Comparing end-user and intelligent remote control interface generation. *Personal and Ubiquitous Computing*, 10(2-3):136–143, 2006.

[163] W.R. Gilks, S. Richardson, and D.J. Spiegelhalter. *Markov chain Monte Carlo in practice*, volume 2. CRC press, 1996.

[164] S.A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the third annual ACM symposium on Theory of computing*, pages 151–158. ACM, 1971.

[165] D. Le Berre and A. Parrain. The Sat4j library, release 2.2 system description. *Journal on Satisfiability, Boolean Modeling and Computation*, 7:59–64, 2010.

[166] E.W. Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

[167] M.L. Fredman and R.E. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the ACM (JACM)*, 34(3):596–615, 1987.

[168] D. Bonino and F. Corno. DogSim: A State Chart Simulator for Domotic Environments. In *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2010 8th IEEE International Conference on*, pages 208–213. IEEE, 2010.

[169] Fulvio Corno and Muhammad Sanaullah. Modeling and Formal Verification of Smart Environments. *Security and Communication Networks*, pages n/a–n/a, 2013.

Chapter 8

Publications

8.1 International Journals

1. Muhammad Sanaullah, Fulvio Corno, Faisal Razzak, (2014) **Automatic Device Activation Regarding User Goals in Smart Environments** In: "Journal of Ambient Intelligence and Smart Environments". pages 23 (Accepted).

2. Fulvio Corno, Muhammad Sanaullah (2013) **Design-Time Formal Verification for Smart Environments: An Exploratory Perspective** In: "Journal of Ambient Intelligence and Humanized Computing", pages 22, DOI: 10.1007/s12652-013-0209-4.

3. Fulvio Corno, Muhammad Sanaullah (2013) **Modeling and Formal Verification of Smart Environments** In: Hangbae C, Lee D, Overill R (ed) Special Issue: Human-centric Security Service and Its Application in Smart Space, "Security and Communication Networks", pages 17, DOI: 10.1002/sec.794.

8.2 Proceedings

1. Fulvio Corno, Muhammad Sanaullah (2011) **Formal Verification of Device State Chart Models** In: IEEE Computer Society (USA), "The 7th International Conference on Intelligent Environments", Nottingham (UK), 25-28 July, pp.66 to 73, ISBN: 9780769544526, DOI:10.1109/IE.2011.36.

2. Fulvio Corno, Muhammad Sanaullah (2011) **Design time Methodology for the Formal Verification of Intelligent Domotic Environments** In: Ambient Intelligence–Software and Applications, Springer Berlin (DEU), "International Symposium on Ambient Intelligence", Salamanca (ES) 6 - 8 April, vol. 92, pp. 9 to 16, ISBN: 9783642199363, DOI:10.1007/978-3-642-19937-0_2